

HEROTALK - A TEXT-TO-SPEECH PHONEME
TRANSLATOR FOR THE VOTRAX SC-01
SPEECH SYNTHESIZER

A Paper
Submitted to the Graduate faculty
of the
North Dakota State University
of Agriculture and Applied Science

by


John Alan Abraham

In Partial Fulfillment of the Requirements
for the Degree of
Master of Science

Major Department: Computer Science

June 1989

Fargo, North Dakota



North Dakota State University

Graduate School

Title

HEROTALK - A TEXT-TO-SPEECH PHONEME TRANSLATOR

FOR THE VOTRAX SC-01 SPEECH SYNTHESIZER

By

John Alan Abraham

The Supervisory Committee certifies that this **paper** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Kenneth Magel

Chair

William Perrizo

Lou Richardson

Kenneth Magel
William Perrizo
Lou Richardson

Approved by Department Chair:

10/26/89

Date

[Signature]

Signature

Abstract

Text-to-speech phoneme translators translate text from an orthographic format into a phonetic format usable by a phoneme-based speech synthesizer to generate speech. "HEROTALK," the text-to-speech phoneme translator developed in this thesis, uses "letter-to-sound" rules to convert typewritten English text to Votrax speech phonemes. A two-step translation process is employed; text is first converted to a standard phoneme set and then converted to the phoneme set usable by the Votrax SC-01 speech synthesizer. This two-step process makes the system easily adaptable to virtually any phoneme-based speech synthesizer. The system developed runs on an IBM-PC and was written in Turbo Pascal (version 3.01A).

Two commercial text-to-speech systems, the IBM-PCjr speech attachment and the Macintosh "Smooth Talker" system, are described and compared to the "HEROTALK" system.

Table Of Contents

Chapter I - Overview	1
Chapter II - Human Speech Production	6
Introduction	6
Sound Sources for Human Speech	6
Resonance	9
Chapter III - Speech Synthesis	12
Introduction	12
Advantages	12
Disadvantages	13
Developments in Speech Synthesis Technology	14
Waveform Encoding/Reconstruction	15
Analog Formant Frequency Speech Synthesis	16
Linear Predictive Coded (LPC) Speech	17
Chapter IV - Text-to-Speech Translation	19
Introduction	19
Text-to-Speech Approaches	19
History	20
Text-to-Speech Translation in HEROTALK	22
The Rules Used in the HEROTALK System	23

Converting English Text to IPA Phonemes	27
B String Matching	30
C String Matching	34
A String Matching	35
 Chapter V - The HEROTALK System	 39
Introduction	39
General Information	39
Program Structure	39
Data Structures Used	40
Implementation	41
The HEROTALK Modules	43
Module HEROTALK.COM	43
The Text-to-IPA Translation Rules	43
User Input	45
Text-to-IPA Translation	49
Module EDIT_IPA.CHN	49
The IPA Editor	50
Module FINISHUP.CHN	52
IPA-to-Votrax Translation	53
Upload	58
 Chapter VI - Results And Discussion	 64
Strengths Of The HEROTALK System	64
Limitations Of The HEROTALK System	65

Suggestions For Improvement	66
Comparison of HEROTALK with the IBM-PCjr Speech Attachment	68
Comparison of HEROTALK with the "Smooth Talker" System	70
Testing the HEROTALK System	71
 Bibliography	 73
 Appendix A - Letter-to-Sound Rules Used in HEROTALK	 75
 Appendix B - S-Record Output Format	 83
 Appendix C - HEROTALK Operation Guide	 88

Chapter I

Overview

Text-to-speech phoneme translators convert text from its orthographic (written) format to a phonetic format usable by a speech synthesis unit thereby enabling the production of intelligible synthesized speech from that text. HEROTALK is a microcomputer-based text-to-speech phoneme translator written for use in conjunction with the Heath ET-18 educational robot (HERO). Built into the HERO robot is a Votrax SC-01 phoneme-based speech synthesizer capable of uttering 64 different phonetic sounds. These sounds represent the primitives of English speech, and when strung together in appropriate combinations can be used to generate spoken English words. Unfortunately, HERO cannot, on its own, construct complete words from these phonemes. Outside of a limited number of "canned" sentences prestored in its memory, HERO does not possess the immediate capability of speaking complete English words. To extend HERO's vocabulary beyond these "canned sentences," the user must manually compute the correct phoneme combinations needed in order to speak a given word, and then manually key into HERO a speech program instructing HERO to speak those phonemes. HEROTALK was written to relieve the user of this task and not only automatically converts typed English text into equivalent strings of Votrax speech phonemes, but also generates the necessary robot code to command HERO to speak this text, loading that code into HERO, ready for execution.

All that is required of the user is to provide HEROTALK with the English text corresponding to the words desired to be spoken.

In writing HEROTALK the following criteria were used as design guidelines. First, I wanted HEROTALK to be capable of translating almost any English text it could receive as input. As the main benefit of using a phoneme-based speech synthesizer is the capability of unlimited vocabulary, HEROTALK's applicability as a text-to-speech translator would certainly be limited if it were restricted in the text it could translate. Second, I wanted the system to be independent of any single speech synthesis unit. By incorporating this independence, HEROTALK can be used with virtually any external speech synthesis unit which generates phonetic synthesized speech. Third, as I wanted the system to be adaptable and modifiable to meet the needs of the user, I wanted to write the system in a computer language which would permit and encourage easy program maintenance and modification. Finally, keeping in perspective the educational premise underlying the HERO robot, I wanted to allow the user to interact with the text-to-speech translation and allow him, prior to the Votrax translation and subsequent generation of speech, the opportunity to review and/or edit HEROTALK's phoneme translation. Through interacting with the text-to-speech translation, not only is the accuracy of the translation improved, but the user of HEROTALK will gain insight as to how English words can be broken down into individual speech phonemes, and how those phonemes can be subse-

quently used to generate computerized English speech.

HEROTALK was written in Turbo Pascal [21] for use on the IBM-PC or compatible. The IBM-PC was chosen because of its widespread popularity and availability. Because of this popularity, a large number of external options are available for use with the IBM-PC including several external phoneme-based speech synthesis units [3, 4, 20]. Similarly, by choosing Turbo Pascal, a popular pascal compiler, as the language in which to develop HEROTALK, future modification is not only facilitated but encouraged.

Basically, HEROTALK accepts English text as input and returns the Votrax phonetic counterpart. Initially, English text is converted into a standard set of speech phonemes -- the International Phonetic Association (IPA) set [7]. Next, the IPA translation is converted into the phoneme set corresponding to the Votrax SC-01 speech synthesizer. Both translations are saved on disk for the user. The text-to-speech phoneme translation itself employs a rule-based system consisting of several "English-to-IPA" translation rules. Each rule consists of a condition-specifying "if" part, corresponding to a character, or combination of characters, of English text (a grapheme), and an action-specifying "then" part, corresponding to the IPA translation of the grapheme specified in the "if" part. A sample of English text is input to HEROTALK by the user, and is examined from beginning to end, one character at a time. As individual characters and character combinations of the English text are encountered in "if" conditions their corresponding

"then" actions (i.e., their IPA translations) are accumulated, the end result being the IPA translation of the aforementioned sample of English text. Once computed, the IPA translation is easily converted to the Votrax phoneme set by merely substituting for each IPA phoneme its Votrax equivalent whereupon the phonemes are then uploaded via the PC's serial communications port to the HERO robot ready for speech production.

HEROTALK is designed to have several advantageous features. First, the translation rule set can be easily modified and/or expanded. Consequently, inadequacies in the rule set can be easily corrected and even new rules added as the user deems necessary. Second, because of the two-step translation process (i.e., translating from English text to IPA phonemes, and then from IPA phonemes to Votrax phonemes), the system with a minimal amount of alteration can be modified to function with virtually any phoneme-based speech synthesis unit. All that would be required is to modify the program code and replace the Votrax phoneme set with the set applicable to the speech synthesizer to be used. Finally, HEROTALK has the capability of handling unlimited vocabulary and will translate almost any English text it is given as input.

The major disadvantage with HEROTALK as it currently exists is its "batch" operation mode. Because of the requirements of the HERO robot, and the data format required to use its speech synthesis unit, HEROTALK was written to operate in "batch" mode, rather than the more desirable "interactive" mode which most other text-to-speech translation programs operate

under. The program could be easily modified, however, to function in an interactive environment.

The remainder of this thesis comprises six chapters. Chapter II temporarily leaves the area of computer speech synthesis to present a brief discussion of the human speech process. Phoneme-based speech synthesis is very analogous to the production of human speech, and an understanding of the latter, I feel, is beneficial in understanding the former. Chapter III looks at speech synthesis examining its advantages and disadvantages, and current developments. Phoneme-based speech synthesis will be discussed along with brief discussions of two other contemporary speech synthesis methods: waveform encoded/reconstructed speech, and linear predictive coded (LPC) speech. Chapter IV addresses text-to-speech translation, reviewing its history and describing in detail the "rule-by-synthesis" method used by HEROTALK to convert English text to IPA phonemes. Chapter V examines the HEROTALK system in detail concentrating specifically on its program structure and major features. Finally, chapter VI discusses the results of testing HEROTALK, compares it with other speech synthesis units of similar nature, and address HEROTALK's strengths and limitations. Appendices include a listing of the "English-to-IPA" translation rules used by HEROTALK, and a HEROTALK operator's guide.

Chapter II

Human Speech Production

Introduction

The paper begins with a brief overview of the area of speech synthesis starting, because of its influential nature in shaping this area, with a discussion of the human speech process. Not only has human speech provided the model for most evolutionary developments in the field of synthesized speech, but it also provides the model for the "formant frequency" method of speech synthesis important in this paper as it is the method utilized by most phoneme-based speech synthesizers including the Votrax SC-01. A more detailed look at speech synthesis is deferred until chapter III. This chapter will focus its attention on the production of human speech, identifying when appropriate analogies between it and the Votrax SC-01.

Sound Sources for Human Speech

Fig. 2-1 shows a simplified diagram of the human speech organs, while Fig. 2-2 shows a block diagram of the human speech system [20]. Human speech is produced by making use of the lungs, throat, mouth, and nose. The lungs serve as the basic source of acoustic power, forcing out a stream of air which is exhaled via the trachea, and through the area from the larynx to the lips known as the vocal tract. Basically, there

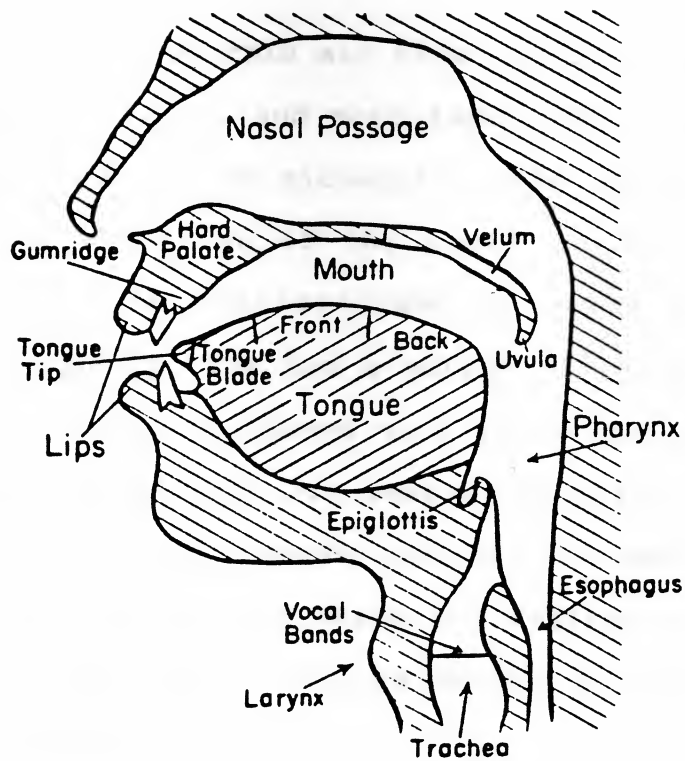


Fig. 2-1 - The Human Speech Organs

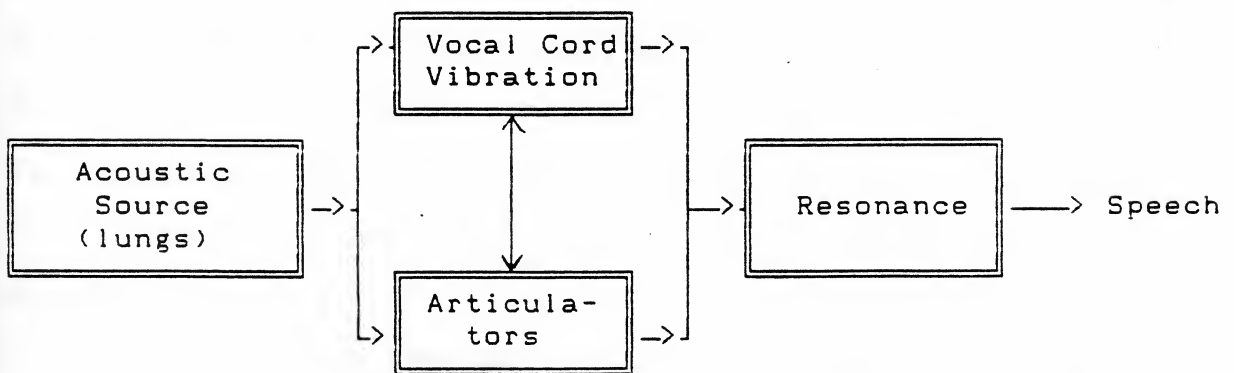


Fig. 2-2 - Block Diagram of Human Speech Process

are three ways in which this air flow can be converted into an acoustic signal. First, and most important, the air stream vibrates the vocal cords situated in the larynx. The sound produced by the vibration of the vocal cords, known as the "glottal pulse," is characterized as being composed of a fundamental frequency tone and a series of harmonics of that tone [2, 8]. Sounds produced with vocal cord vibration are termed "voiced" sounds and are best exemplified in the vowel sounds and in consonant sounds such as "r" and "l." If the air flow is diverted at the velum and redirected through the nasal cavity, the class of voiced speech sounds known as the "nasals" are produced [2, 8].

Second, using articulators such as the lips, tongue, or teeth, the air flow can be constricted somewhere in the vocal tract, thereby generating an "air turbulence." Sounds formed in this way are generally of high frequency composition and characterized by a "breathy" or "hissy" quality. Examples include consonant sounds such as "s" and "f," and whispered vowels. Sounds created through turbulence are also referred to as "fricatives." Some speech sounds, in particular "voiced fricatives" such as "v" and "z," are produced by using both air turbulence and vocal cord vibration together. Sounds not incorporating vocal cord vibration are termed "unvoiced" sounds.

Finally, using the same articulators, the vocal tract can be closed at some point thereby letting air pressure build-up behind the closure. Upon reopening the tract, the sudden

burst of air which emanates from the mouth results in a "plosive" sound. The English dialect contains both voiced and unvoiced "plosives." Examples include "b," "d," "p," "t," and "k."

Comparatively, the Votrax SC-01 uses two electronic noise generators as sound sources for its speech production. One generates an oscillating tone closely resembling the human glottal pulse and is used in the synthesis of voiced sounds. A second simulates the effect of "air turbulence" and generates a fricative noise used in producing sounds with fricative characteristics. Plosives are generated using a combination of these two noise generators. From these noise generators, the Votrax SC-01 can generate the spectrum of phonetic sounds necessary to adequately reproduce English speech [3, 22].

Resonance

A major contributing factor in forming speech sounds, particularly voiced sounds, is the phenomenon of resonance. Resonance refers to the ability of a vibrating force to drive into vibration a volume of air thereby increasing the power put out by that original vibrating force [8]. In human speech production, the vocal tract acts as a resonant cavity taking voiced and unvoiced sounds, and amplifying and attenuating specific frequencies in them to generate the distinct acoustic sounds which we interpret as speech phonemes. The resonant properties of the vocal tract depend mainly on its dimensions and detailed shape at any particular moment, with the tongue,

lips, jaw, and velum the principal agents involved in altering these dimensions. As they change, the tract will respond to frequencies present in any given sound whose wave-lengths happen to fit with its dimensions. The result is that in the sound which finally emanates from the speaker's mouth any of the harmonics which respond to the resonances of the tract will be relatively strong while those that do not will be relatively weak. This explains when speaking the constant movement of the vocal tract's articulators, for each unique phonetic sound correspondingly requires a unique vocal tract dimension for its production. Though the vocal tract, as is the case with any resonating system, will generate several resonances, for the purpose of speech only its first three or four are essential. A special term is used to refer to these resonance frequencies; they are called "formants," and are labeled by numbering them from the lowest resonance frequency upwards. Each formant corresponds to a particular frequency range in the audio spectrum, with the exact placement of the formant frequencies within this spectrum instrumental in determining the speech sound (phoneme) generated. For example, the long "e" sound, as in the word "feet," can be adequately reproduced with frequency tones of 250 Hz (formant one), 2300 Hz (formant two), and 3000 Hz (formant three) [3]. Most of the distinctions on which the English language depend are carried by the first three formants with formants one and two making the greatest contribution.

In the case of unvoiced sounds, the sound generated is not

composed of a fundamental frequency and harmonics, but instead contains all frequencies (i.e., "white" noise). In this case it is more appropriate to think of the vocal tract as having a "filtering" effect on the noise. The speech system generates basically a white noise with the effect of the vocal tract being to filter out those frequencies nonessential to the desired sound [2, 8].

In comparison, the Votrax SC-01 simulates resonance through the use of four adjustable bandpass filters each corresponding to one of the first four "essential" formants present in English speech. Whenever a specific speech phoneme is to be generated, the SC-01, using data stored in "phoneme control tables," first identifies the correct combination of frequencies needed to generate the phoneme in question. The frequency of each filter is then adjusted to match an equivalent required resonance frequency of the vocal tract during human speech. Upon completing these adjustments, the SC-01's noise generator(s) send(s) the appropriate "noises" through these filters, all unnecessary frequencies are attenuated, and the resultant sound, generated from the summation of the output of the four bandpass filters closely resembles human speech [3, 22].

Chapter III

Speech Synthesis

Introduction

Speech synthesis refers to the ability to create spoken words, and speech signals, by artificial means. Current applications include talking typewriters, educational toys, and automobile warning systems as well as numerous devices designed to aid those handicapped by voice and/or vision impairment. In this chapter the area of speech synthesis is addressed, beginning with a brief discussion of its advantages and disadvantages and concluding with an examination of its current state, including a comparison and contrasting of today's three major approaches to computer speech synthesis.

Advantages

An obvious advantage of speech synthesis is, of course, the enrichment it can add to the lives of the handicapped. Because of synthesized speech, those handicapped with voice and/or vision impairment can now co-exist in environments previously inaccessible to them due to their lack of communicative skills. Already for those handicapped by vision impairment talking reading machines have been developed [15]. Using the technology of optical character recognition, combined with unlimited vocabulary synthesis, vision impaired individuals can now enjoy the benefits of reading. Similarly for those handicapped by loss of speech speaking machines have been developed allowing users to speak with an artificial voice almost as

easily as they can point to a word [3, 24].

From an educational perspective, speech, when used in conjunction with computer-aided instructive tools, can be used to clarify instructions and/or emphasize important ideas and concepts. Scholastic's "Talking Text Writer" and IBM's "Writing to Read" are representative of two applications where speech synthesis has been successfully applied in an educational setting. In these cases, speech synthesis is used to teach reading and writing [1]. Whereas computer aided instruction is often criticized as being cold and impersonal, with the addition of speech computers become more "friendly" and less impersonal.

Finally, when used as a warning device speech presents an interruption which demands attention. Whereas a flashing light or a written message appearing on a screen might be inadvertently missed, a spoken warning will not. Furthermore, the knowledge that warnings and/or errors will be spoken frees the operator of a device from monitoring a display screen or a panel of warning lights thereby enabling him to devote full attention to the matter at hand -- operating the device.

Disadvantages

One of the major disadvantages inherent in the use of synthesized speech is the very fact that it is just that--artificially produced speech. Consequently, situations will arise when what is spoken by the synthesizer will not be understood. Such a possibility necessitates the need for alterna-

tive, conventional means of reviewing spoken messages and information. Also, despite its potential as an "attention grabber," with time a user may become too accustomed to a particular tone of voice. Thus, just as we may "tune-out" an individual speaking to us, we may also, in time, learn to ignore a particular synthesized voice. In the case of a warning, the results may be disastrous. A possible solution, perhaps, would be to vary the tone of the synthesized voice thereby eliminating the listener's expectation of its sound. One final disadvantage is the high cost of speech synthesis units. Unfortunately, the higher the quality of speech synthesis, the more costly the initial investment. Fortunately, as speech synthesis technology improves prices will gradually decrease thereby making units of considerable quality affordable.

Developments in Speech Synthesis Technology

Interest in speech synthesis can be documented as far back as the eighteenth century. Synthesizers made in those "pre-electronic" days were basically acoustic models of the vocal tract, with most all of these developments focusing on the production of speech sounds (phonemes) [2, 3, 15, 24].

The majority of research in speech synthesis during the last fifty years has concentrated mainly on computer-generated speech and basically falls under one of three major methods of speech synthesis. These are in order of increasing complexity the waveform encoder for direct speech reconstruction, the

analog formant frequency synthesizer for phonetic speech synthesis (the method employed by the Votrax SC-01 speech synthesizer used in the HERO robot), and mathematical reconstruction of speech from its various frequencies and vocal characteristics, the method employed in linear predictive coded (LPC) speech synthesis [3].

Waveform Encoding/Reconstruction

Waveform encoding/reconstruction of speech is one of the most basic and elementary forms of computerized speech synthesis. Operationally, the computer becomes in effect a simple voice recorder. The desired output phrases and words are digitally stored as response units and "played back" by application programs as needed. One of its most prevalent attributes is its clear, intelligible speech. However, while the fidelity of its spoken output is extremely clear, the memory requirements for this output can be overwhelming. As an example, the IBM-PCjr Speech Attachment, a waveform encoded speech synthesizer for use with the IBM-PCjr, requires nearly 24K bytes of memory to store a mere 5 seconds of digitized speech [11]. This presents the major drawback with the waveform encoded method -- its inefficient use of memory. Because each word or phrase must be digitally sampled and stored, large amounts of memory are consumed. Furthermore, because its speech capability is limited to its prestored vocabulary, the capability for generating a word that has not been "prestored" does not exist. Consequently, it is often used in applications

where a small vocabulary is required [3].

Analog Formant Frequency Speech Synthesis

The formant frequency synthesis method is particularly relevant to this paper as it is the method employed by the HERO robot's Votrax SC-01 speech synthesizer. Formant frequency voice synthesizers are typically rather "robotic" in sound mainly due to their lack of direct human speech input for their resultant spoken output. In other words, speech synthesized by a formant synthesizer is truly originated by the computer. The principles behind the formant synthesizer are based on acoustic replication of the human vocal tract. Noises are generated using voiced and unvoiced noise generators with bandpass filters used to create formant frequencies from these noises characteristic of human speech. The sum of these formant filters because it so closely resembles the frequency spectrum of human speech is interpreted by our ears as human in sound. By actually generating speech sounds, as opposed to storing complete digitized words in memory, the formant frequency synthesizer is much more memory efficient than waveform encoding.

The most common method for creating connected speech with the formant frequency synthesizer is through the use of numerous identifiable speech sounds called "phonemes." By stringing together appropriate combinations of phonemes the synthesizer can create spoken words closely replicating those of human speech. Consequently, an additional advantageous feature of

the formant frequency method is its inherent unlimited vocabulary capability. Since speech is generated by the re-creation of individual speech sounds, any word can thus be spoken by correctly piecing together its required phonetic sounds. Formant frequency synthesizers are often used, consequently, in conjunction with "text-to-speech" translators which automatically convert text into its correct phonetic structure. Nonetheless, the formant synthesizer is not always easily understood. Unfortunately, the great number of English language exceptions to any general rules relating text and spoken words creates problems with generating an unlimited vocabulary. Text-to-speech phoneme translators greatly help this task, but as a rule of thumb you can generally speak only one or two sentences from a text-to-speech synthesizer before the system runs across a word that it incorrectly parses resulting in unintelligible speech. This is not really a fault of the phoneme-based speech synthesizer, however, but rather a result of the numerous inconsistencies in the rules of English speech [3, 4, 7, 12].

Linear Predictive Coded (LPC) Speech

The third major type of speech synthesis can best be described as a digital modeling of the human vocal tract. Synthesis methods of this type are highly mathematical in their operation because they simulate the actions of the human vocal tract with mathematical equations. Currently, the most typical application of this technique is linear predictive coded (LPC)

speech.

The vocabulary for LPC speech as in waveform encoding is typically generated by a human speaker. However, rather than directly storing the spoken vocabulary as sampled information, the speech is first digitally analyzed separating it into its various voice characteristics such as pitch, amplitude, and formant frequency composition. These characteristics are then converted and stored in the form of numerical coefficients which can later be used to mathematically regenerate the original speech signal. Though the theory behind this method of speech synthesis seems very similar to that of waveform encoding, the primary difference lies in the amount of speech storage available for the same amount of memory. Compared to our previous example involving the IBM-PCjr Speech Attachment, the Hitachi HD61885 Speech Synthesizer (an LPC based unit) contains a 4K byte ROM chip capable of storing 26 seconds of speech [3]. Obviously, the aforementioned "speech dissection" greatly decreases the amount of storage required for its spoken output. The result is a lower cost per stored word in the final product [2, 3, 24].

When the speech is later regenerated, its sound is very close to normal human speech. It thus blends the memory efficiency of the formant frequency method with the clarity and intelligibility of the waveform encoder/decoder. It does, however, suffer the drawback of being limited to a prestored vocabulary.

Chapter IV

Text-to-Speech Translation

Introduction

In this and the next chapter the HEROTALK text-to-speech system will be discussed. This chapter places emphasis on text-to-speech translation including an overview of text-to-speech translation, its historical development, and an in-depth analysis of HEROTALK's algorithm for converting English text into IPA speech phonemes.

Text-to-Speech Approaches

Basically there exist two approaches to solving the problem of converting text into speech phonemes. The first involves the use of a pronunciation dictionary consisting of numerous English words and their respective phonetic translations. The algorithm employed by the text-to-speech system isolates individual words in the text to be translated, and looks up each isolated word in the pronunciation dictionary. Unfortunately, a translator based on this approach is only as encompassing as its dictionary. Because every word appearing in the text to be translated must be included in the dictionary, the dictionary can grow to enormous sizes, ultimately consuming much of the computer's available memory. Obviously, regardless of how large we allow the dictionary to grow, it cannot possibly store a phonetic translation for every word in the English language. Consequently, such an approach lacks the

capability of unlimited vocabulary translation.

A second approach uses grapheme-to-phoneme translation rules. Such rules attempt to describe a correspondence between the orthographic and phonetic forms of a language and are used to best estimate the pronunciation of a given word. Each rule specifies a phonetic correspondence to one or more letters of text, with a given letter's context often used to determine which rules should be applied. The benefit of this approach is that any input text can be translated, thereby giving such a system the capability of unlimited vocabulary translation. Furthermore, the size of such a rule set is generally much smaller than a "look-up" dictionary and thereby much more memory efficient. Nonetheless, the translation can only be as accurate as the translation rules employed, and the numerous inconsistencies between orthographic and spoken words make it difficult to create an encompassing set of rules which can take into effect every possible exception and inconsistency.

History

The first text-to-speech system was described by Teranishi & Umeda in 1968 and based on a 1500 word "look-up" dictionary [15]. Associated with each word was, among other attributes, its phonetic pronunciation. Since then, text-to-speech systems have been developed ranging in complexity from letter-to-sound rule systems to combined dictionary look-up/letter-to-sound rule systems with syntactic analysis [12]. Among the latter are systems developed by Ainsworth (1973) and McIlroy (1974)

[12, 15]. In these two systems, words input to the system are first checked against a list of exceptions. If they are not on the list, "letter-to-sound" rules are then applied which take into account each letter and its context.

The "text-to-speech" system developed by Allen (1973) is more ambitious [15]. Allen proposed a system combining rules with a dictionary of word roots, or "morphs." Every word of input text is regarded as either a morph, a compound morph, or a morph plus affix (suffix or prefix). The premise underlying the system is that prefixes and suffixes can be stripped away from any complex word revealing its essential morph structure. An input word is first presented to the morph dictionary and if not listed is reduced, if possible, to a smaller morph and that reduced word then checked against the list. This sequence is repeated until the reduced word can be identified as a morph. In the event the "word reduction algorithm" fails, letter-to-sound rules are then applied to "guess" at the word's pronunciation.

The "Klattalk" system developed by Klatt (1982) represents one of the most elaborate text-to-speech systems [12]. Using letter-to-sound rules, not only does the system convert text to speech phonemes but encompasses within that phonetic conversion information denoting lexical stress, pitch control, and other syntactical enhancements.

In 1976, Elovitz, Johnson, McHugh, and Shore developed a small set of rules for translating English text to speech phonemes [7]. Stemming from a desire to avoid dependency on a

particular speech synthesizer, their system translated text into a standard set of phonemes, the set of the International Phonetic Association (IPA). Because the text-to-phonetics information is contained in device independent rules, systems designed around these rules are not dependent on any single speech synthesizer.

Text-to-Speech Translation in HEROTALK

HEROTALK is a microcomputer-based text-to-speech system developed for use with the Heathkit ET-18 educational robot (HERO). Built into the HERO robot is a Votrax SC-01 phoneme-based speech synthesizer capable of uttering 64 different phonetic sounds. HEROTALK takes English text as input, and converts that text into Votrax speech phonemes. HEROTALK was written for use on the IBM-PC and interfaces with the HERO robot via the PC's serial port. Because of this separation, one of HEROTALK's major design requirements is the production of a system which can be used with virtually any speech unit. A second design requirement was the desire to produce a system capable of unlimited vocabulary translation. This seemed only natural since we are dealing with phoneme-based speech, and phoneme-based speech synthesizers. Finally, a third requirement was to keep the rule set as small as possible. Because microcomputers are often limited in memory, the speech system itself cannot consume too much of the available resources. This also ruled out the possibility of using a "look-up" dictionary of any surmountable size.

The text-to-speech translation algorithm for HEROTALK uses a set of letter-to-sound rules to convert English text into speech phonemes. The "letter-to-sound" rules developed by Elovitz, et. al. are used as a foundation upon which to build the HEROTALK system. Not only do they constitute a small set, but they also translate text into a machine independent phonetic set, that of the IPA shown in Table 4-1.

The Rules Used in the HEROTALK System

The translation rules used by HEROTALK consist of ASCII character strings, and for use by the system are stored in the external system file "RULES.DAT." The complete list of "text-to-IPA" rules used by the HEROTALK system is listed in Appendix A. The format of the rules is:

RULE STRING = RULE 1, RULE 2, ..., RULE N

with each rule of the form:

$$A[B]C=/D/$$

which means that string B of English text, preceded by string A and followed by string C of English text, is translated as string D in IPA phonemes. The rules are grouped together, in file "RULES.DAT," according to the first character of each rule's B string. For example, all the rules in which B strings begin with the letter "A" belong to the "A rules." Similar rule groups are constructed for the remaining letters of the alphabet, as well as a separate group for numerical characters,

AA - <u>f</u> ather	G - <u>g</u> un	SH - <u>s</u> hip
AE - <u>f</u> at	HH - <u>h</u> ow	T - <u>t</u> ime
AH - <u>b</u> ut	IH - <u>b</u> it	TH - <u>e</u> ther
AO - <u>l</u> awn	IY - <u>b</u> eet	UH - <u>f</u> ull
AW - <u>h</u> ow	JH - <u>j</u> ar	UW - <u>f</u> ool
AX - <u>a</u> bout	K - <u>c</u> an	V - <u>v</u> ault
AY - <u>h</u> ide	L - <u>l</u> augh	W - <u>w</u> as
B - <u>b</u> ack	M - <u>m</u> atch	WH - <u>w</u> here
CH - <u>c</u> hair	N - <u>n</u> ow	Y - <u>y</u> oung
D - <u>d</u> ime	NX - <u>s</u> ing	Z - <u>z</u> oo
DH - <u>t</u> his	OW - <u>l</u> one	ZH - <u>l</u> eisure
EH - <u>g</u> et	OY - <u>b</u> oy	* - pause
ER - <u>m</u> ur <u>d</u> er	P - <u>p</u> ack	\$ - pause
EY - <u>g</u> ate	R - <u>r</u> ate	~ - pause
F - <u>f</u> ire	S - <u>s</u> un	

"*" denotes a "long pause"; a sentence terminator for example (i.e., period, etc.)

"\$" denotes a "short pause"; a comma for example

"~" denotes a "brief pause", and is used between words in the IPA translation

Table 4-1 - The IPA Phoneme Set

and one for nonalphanumeric characters (e.g., "&" and "%"). In this way, a particular set of rules can be quickly identified without having to search through the entire set of "letter-to-sound" rules.

Because the rule strings are repeatedly accessed during the translation process they are stored in the PC's RAM, enabling the fastest access possible during text-to-IPA translation. Doubly linked lists are used because they allow traversal of the rule strings in either direction, a necessary feature which will be clarified when we examine the translation algorithms. The rules in each particular rule string are arranged in such a manner that rules with larger B strings precede rules with smaller B strings. This ensures that the text characters being translated are always translated in the largest groups possible. Furthermore, the translation rules in each string are arranged in such a way that the translation algorithm can enter the rule string for a particular letter at only one point -- the head of the list. This gives us control over the order in which the rules in each rule string are evaluated during translation.

In each particular rule, only the B string corresponds exactly to the text being translated. The D strings consist of IPA symbols, whereas A and C strings represent patterns of English text appearing as context for the B string and can contain special symbols which denote special character classes or combinations. These special cases are summarized in Table 4-2 [7]. Individual rules are separated by means of the "/"

- # - One or more vowels
- . - One of B, D, V, G, J, L, M, N, R, W, or Z (a voiced consonant)
- % - One of ER, E, ES, ED, ING, or ELY (a suffix)
- & - One of S, C, G, Z, X, J, CH or SH (a fricative sound)
- @ - One of T, S, R, D, L, Z, N, J, TH, CH, or SH (a consonant influencing the sound of a following U, as in "rule" or "mule")
- ^ - One consonant
- + - One of E, I, or Y
- : - Zero or more consonants

Vowels are: A, E, I, O, U, Y

Consonants are: B, C, D, F, G, H, J, L, M, N, P, Q, R, S, T, V, W, X, Z

Table 4-2 - Special Symbols Appearing in the English to IPA Translation Rules

character, and each rule's D string is delimited by "/"'s"-- one at the front of the D string, and one at the end. By keeping track of the number of "/"'s" encountered in each individual rule string, we are able to keep track of different rules within each string. Further discussion regarding the rule set, including the algorithms used for processing and storing it in RAM, is deferred until Chapter V.

Converting English Text to IPA Phonemes

The algorithms employed in HEROTALK for the translation of text-to-IPA are adapted from work done by O.R. Omotayo. In his paper regarding the conversion of text into speech [18], he suggests a strategy for the development of a microcomputer-based text-to-speech system. HEROTALK represents modifications and enhancements to this design necessary to meet the guidelines set forth earlier in this paper, as well as the requirements of the HERO robot.

The HEROTALK system is comprised of three major modules: "HEROTALK.COM," the major module of the system, "EDIT_IPA.CHN," the IPA modification module, and "FINISH-UP.CHN," the module which completes the speech process by sending the phonemes generated by the system to the HERO robot. The translation process discussed in this chapter is part of the HEROTALK.COM module. Because of the importance of the "text-to-IPA" translation process it will be discussed separately in this chapter. An encompassing discussion of the entire HEROTALK system follows in Chapter V.

For the remaining discussion in this chapter, it is assumed that the HEROTALK system has processed a sample of English text for translation. Although text input will be addressed in Chapter V, it is pointed out here that as with our translation rules, our sample of text is stored in RAM in the form of a doubly linked list. This, as is the case with the translation rules, enables us to traverse the list of input text in either direction.

Once our text has been readied for translation, it is passed to a translate routine which converts English text to IPA phonemes. Fig. 4-1(a) shows a flowchart of the English-to-IPA translation algorithm. Beginning with the first character of the list, the algorithm scans the text from beginning to end and for each character encountered searches the rules pertinent to that character until it finds a rule whose A, B, and C strings match those of the text being translated. The D string of the matched rule represents the IPA translation of its corresponding B string. Similarly, its B string corresponds to a character or group of characters in our list of input text. The D string is copied to an additional linked list and the characters in our list of text corresponding to the characters in the B string skipped over. With the translator now positioned at the next block of untranslated characters in our text the algorithm repeats, continuing until all text has been translated. Upon reaching the end of our text we have its equivalent in IPA phonemes.

The translation process itself is composed of three major

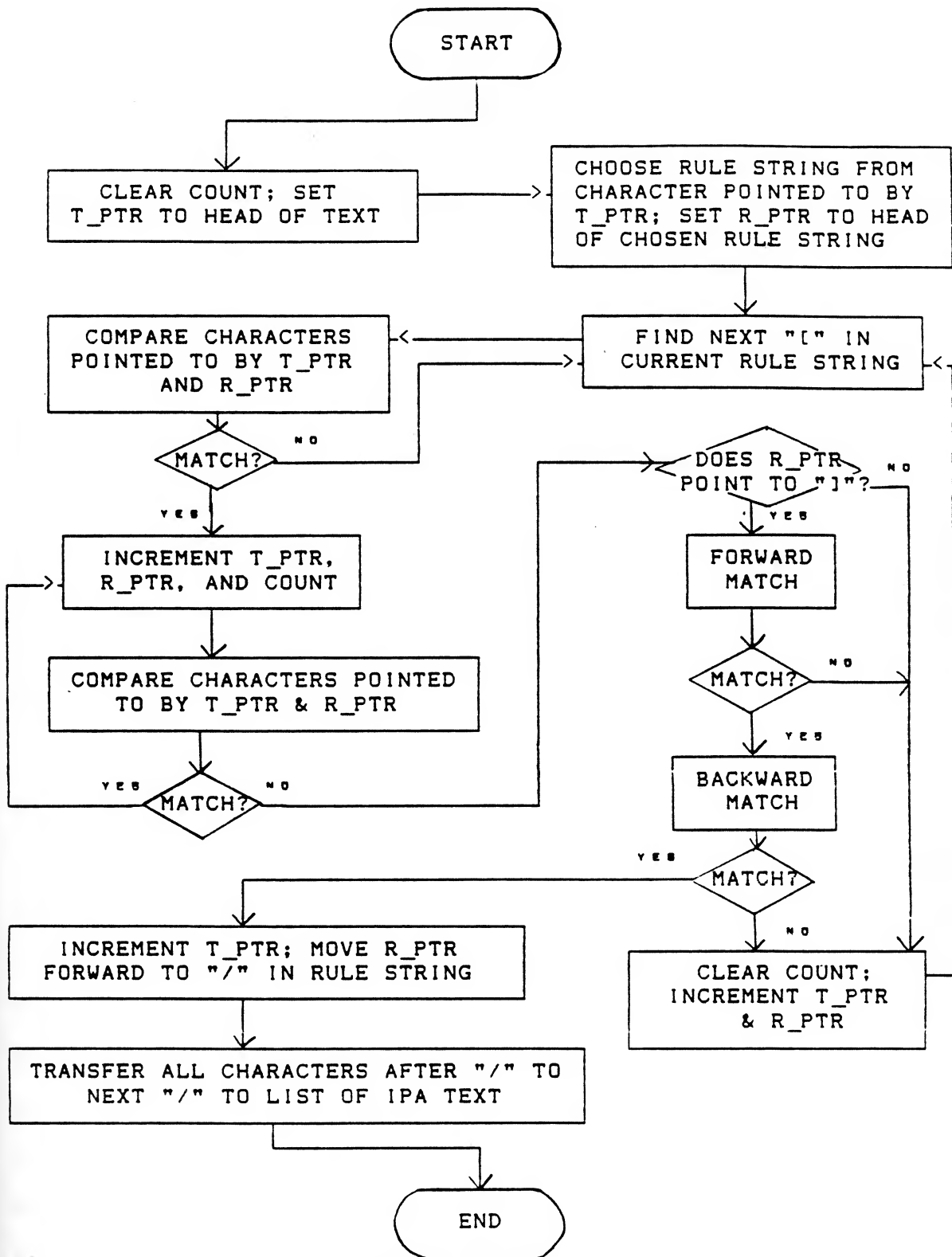


Fig. 4-1(a) - Flowchart for "text-to-IPA" Conversion

sub-algorithms: one which matches our text with rule B strings, one which matches our text's "post" context with rule C strings, and one which matches our text's "pre" context with rule A strings. B string matching initiates an attempted rule comparison, followed by, if successful, an attempted C string match using the same rule. If the C string match fails, we move to the next rule in the rule string, and begin again with an attempted B string match. On the other hand, if the C string match is successful, we then follow it with an attempted A string match using the same rule. These match algorithms, in the source code, are referred to as "translate," "forward matching," and "backward matching" algorithms (B, C, and A string matching respectively). Fig. 4-1(b) shows a flowchart of the "forward matching" algorithm, and Fig. 4-1(c) shows a flowchart for the "backward matching" algorithm.

B String Matching

When text-to-IPA translation begins two pointers are created which monitor the translation. One, "text_ptr," is initially set to the first non-blank character in our list of text. The second, "rule_ptr," after identifying the character pointed to by "text_ptr", is set to the head of the rule string corresponding to B strings which begin with that character. In the aforementioned flowcharts these variables are referred to as "T_PTR" and "R_PTR" respectively. A third variable, COUNT, is used to monitor the number of characters matched in a given rule's B string, and is initialized to 0. As each separate

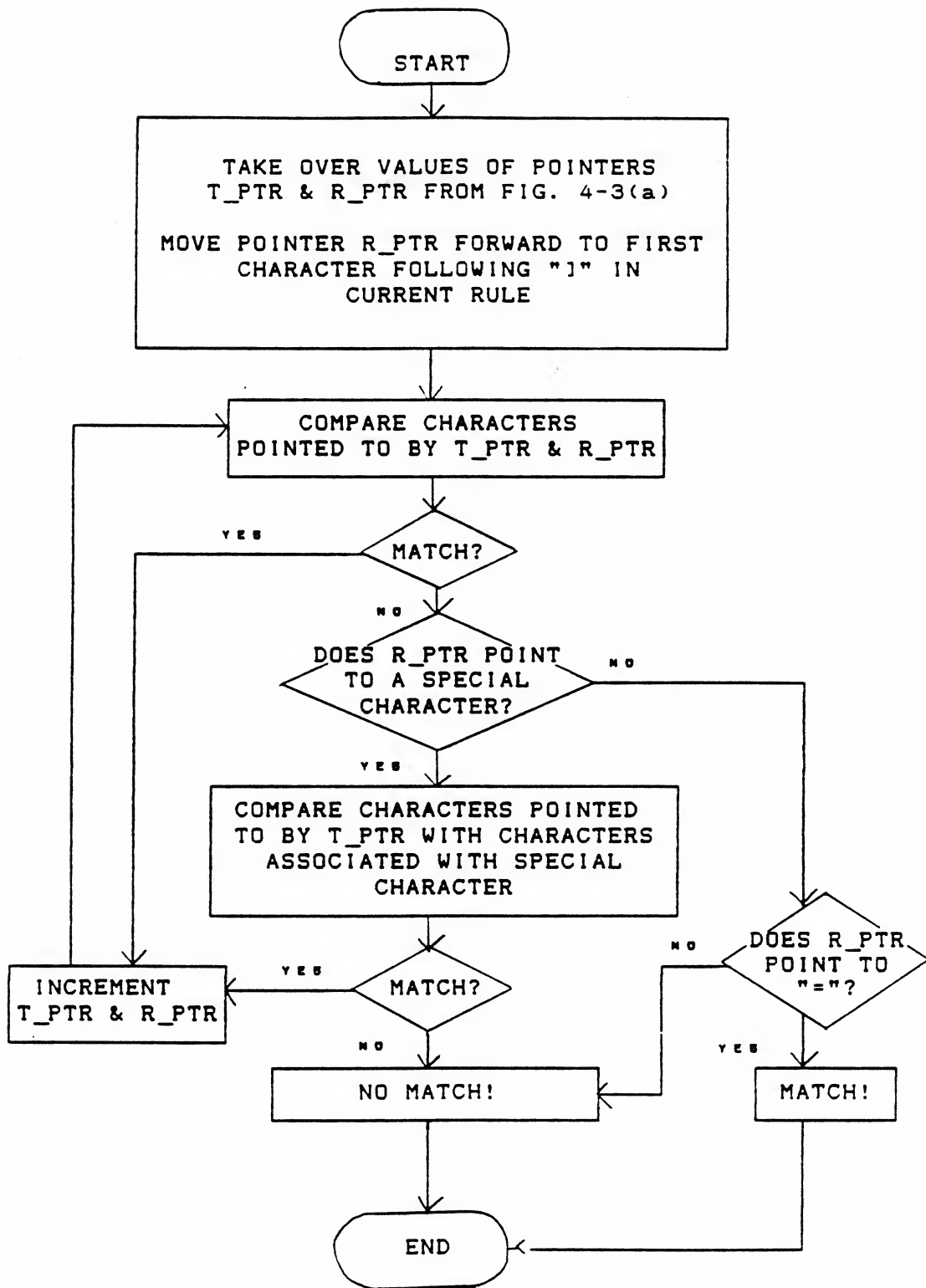


Fig. 4-1(b) - Flowchart for C String (Forward) Matching

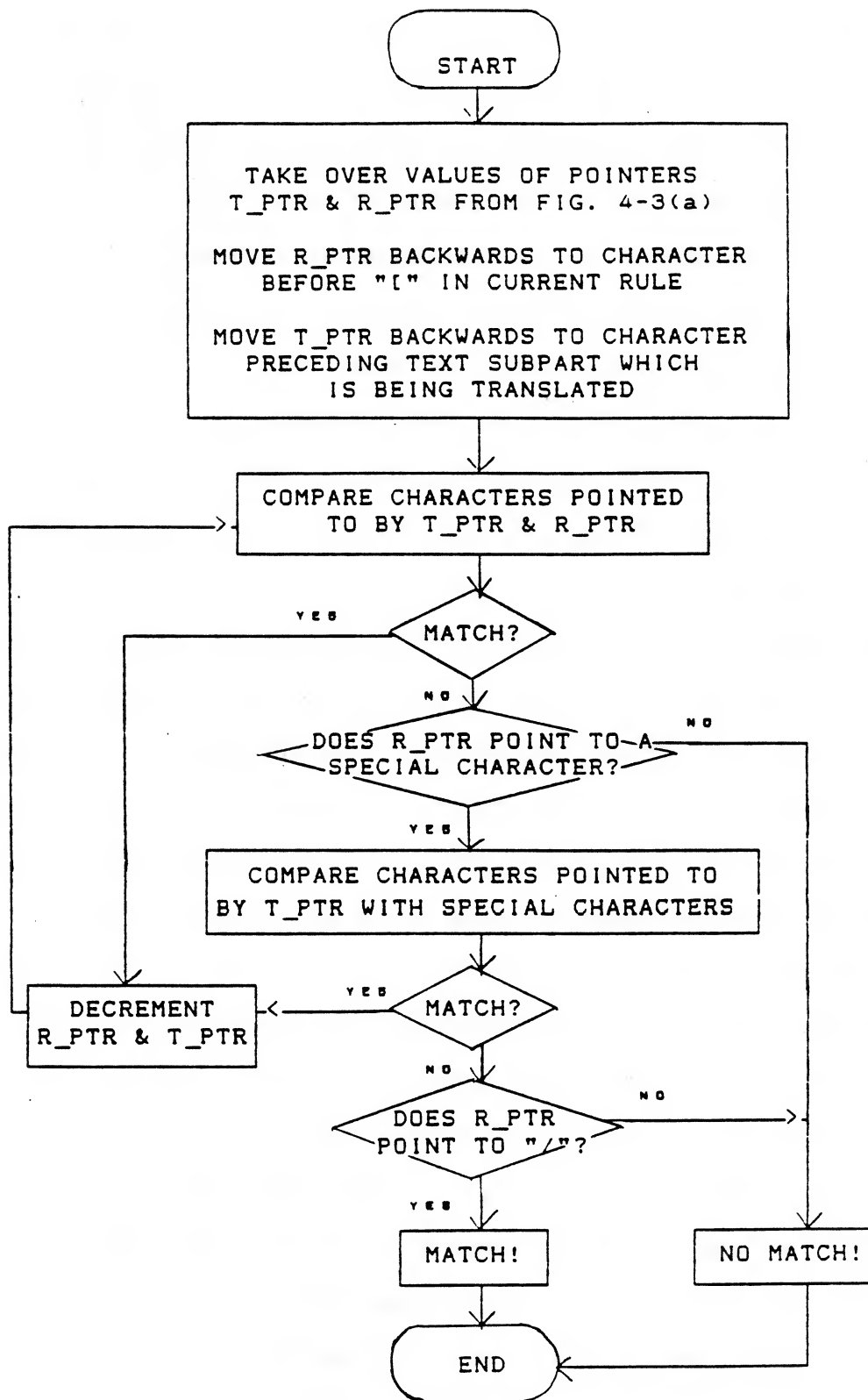


Fig. 4-1(c) - Flowchart for A String (Backward) Matching

rule's B string is delimited with a left bracket character ("["), "rule_ptr" is subsequently moved forward in the selected rule string until the first "[" is located. "Rule_ptr" is then incremented to the next character following the located "[," and thus onto the first character of the B string of the first rule in the chosen rule string. At this point, the characters pointed to by "text_ptr" and "rule_ptr" are compared. If they match, COUNT is incremented and both pointers moved to the next character in their respective lists. These characters are then compared. So long as the characters pointed to by "text_ptr" and "rule_ptr" match, translation using the same rule continues. It should be noted that all lower case characters are temporarily converted to upper case during character comparison. Consequently, a character match will never fail due to "case" differences. If "text_ptr" and "rule_ptr" do not match, the character pointed to by "rule_ptr" is examined. If it is a right bracket ("]"), we know we have reached the end of the B string in the current rule under consideration and thus have located a successful B string match. We then attempt to match the current rule's C string with the text following the text just matched in our list of text (i.e., its "post" context). If "rule_ptr" does not point to a right bracket, we know that this particular rule has failed to match with the text in question. In this case COUNT is reinitialized to 0 and "rule_ptr" is moved forward in the rule string until the next left bracket is located, thereby readying the system for an attempted B string match with the next rule in the current rule

string.

C String Matching

C string matching, or "forward matching," attempts to match the characters in our text following the character(s) matched in the current rule's B string with those of its C string. As forward matching begins, "text_ptr" is already positioned on the first character following the character(s) previously matched with the current rule's B string. Next, "rule_ptr" is moved forward in the rule string to the character following the recently encountered right bracket. This places "rule_ptr" at the beginning of the current rule's C string, ready for comparison to the characters pointed to by "text_ptr." Characters pointed to by "text_ptr" and "rule_ptr" are then compared one-to-one, with each match causing an increment of both pointers to their next respective character and continued comparison. In the event the characters pointed to by "text_ptr" and "rule_ptr" do not match, two possibilities exist. First, "rule_ptr" is examined to determine whether it points to one of the "special characters" mentioned earlier, and detailed in Table 4-2. If "rule_ptr" does point to a character represented in the group of "special characters," the characters pointed to by "text_ptr" are then compared with the characters associated with the special character in question. A separate routine is provided to deal with each of these special cases. If the necessary character matches are found for the special case in question, both

"text_ptr" and "rule_ptr" are incremented, and individual character comparison resumes. If the required character matches are not met, a flag is set denoting the match failure, and "text_ptr" readjusted so that it points to the character pointed to when B string matching was initiated. After "rule_ptr" is moved to the next rule in the rule string, the process begins again with an attempted B string match with the new rule. Second, if "rule_ptr" does not point to a "special character" it is examined to verify whether we have reached the end of the rule in question (i.e., "rule_ptr" points to the character "="). If we have, then all characters of the rule have matched ("=" denotes the separator between the C and D strings). In this case, a flag is set denoting the successful forward (C string) match, and the translation process proceeds to check for a possible A string match with the current rule. If "rule_ptr" does not point to the character "=", then we know the C string match has failed and a flag is set denoting the failed attempt. Translation then resumes with a pointer adjustment of "rule_ptr" to the next rule in the rule string, "text_ptr" back to its original starting point, and another attempted B string match.

A String Matching

The A string matching ("backward matching") routines represent the final check in the scrutinization of a given translation rule. In order to reach this stage of the translation process, a given rule's B and C strings must have already

matched with the text in our English text. A string matching works similar to C string matching except, as the name implies, we move backwards in our respective lists as we compare characters. This clarifies our need for doubly linked lists, for in using doubly linked lists bi-directional traversal is enabled.

Upon initiating A string matching "rule_ptr" is first moved backwards in the current rule string until the character preceding the left bracket, in the current rule, is located. This places "rule_ptr" at the first character preceding the B string in the current rule (i.e., the last character of the rule's A string). In addition, "text_ptr" is moved backwards to the character which precedes the character upon which we initially began matching (i.e., "text_ptr" is positioned at the first character preceding the text actually being translated by the rule in question). Characters pointed to by these pointers are then compared, with each match causing a decrement of each pointer prior to further character comparison. Any "special characters" encountered in the rule's A string are handled as they were with the "forward matching" routines, with the exception, of course, that respective pointers are decremented as opposed to incremented. If during character comparison a character match fails, again two possibilities exist. First, "rule_ptr" is checked to see if it points to the character "/." Such a character denotes that the entire A string has been compared to the characters in our text, and that the A string matches, subsequently meaning that the text corresponding to the B string in this same rule translates as string D in

IPA phonemes. A flag is set denoting that this rule has translated COUNT characters of our English text, and the system saves the D string as part of the eventual output of HEROTALK. "Text_ptr" is then incremented COUNT characters forward from its original starting point and the entire translation process repeats, this time with a new start point, and subsequent character upon which rules are selected. If, on the other hand, "rule_ptr" does not point to the character "/" we know that this rule has failed to meet the requirements for a successful A string match. In this event we return to B string matching with another rule from the current rule string.

Punctuation marks encountered in the English text are translated as they are encountered. Those which have "speakeable" words associated with them (e.g., &, %, etc.) have individual rules defined which are included in the rule string "punc_rules." Those not included in "punc_rules," yet encountered in the English text, represent necessary pauses which are to be incorporated into the IPA translation. A "long" pause is inserted to represent periods, question marks, and exclamation points whereas a "short" pause is inserted to reflect commas, colons, and semicolons. A brief pause is also inserted between each individual word translated. This short pause enhances the aesthetic quality of the speech and helps to distinguish one word from another when speech is produced. Included in Table 4-1 is a summary of the IPA symbols used to represent these various pauses.

It should be pointed out that each rule string contains a

"default" rule which ensures that at least one character of our English text is translated with every new position of "text_ptr" in our list of text. The "default" rules are comprised of a B string consisting of a single character, and an A and C string consisting of no characters at all. Consequently, the system will never encounter a word for which it cannot generate some attempted IPA translation. Default rules are always the last rule in a given rule string.

As might be guessed, the process of finding the correct rule in a given rule string which meets all the necessary criteria can be time consuming. Storing the rules in the form of linked lists, alphabetically according to the first letter in their B strings, greatly helps in that the correct subset of translation rules can be quickly isolated thereby reducing some of the search time. Furthermore, arranging the rules within each rule string such that larger B strings are matched before smaller B strings cuts down on the number of rule considerations required to translate a sample of English text. The larger the blocks of text translated with each successful rule match, the fewer rules we must use in the translation and consequently, the less time we spend searching for correct translation rules thereby decreasing the total time necessary for a complete text-to-speech phoneme translation. Of course, the size of the original English text sample is also influential in determining the translation time. Obviously small samples of text will translate faster than larger samples.

Chapter V

The HEROTALK System

Introduction

The sections in this chapter describe the HEROTALK system. General information about the program's structure, modules, implementation, and employed data structures is given. In addition, important aspects of the system are isolated and examined in detail. These aspects include rule processing, user input, editing the IPA translation, IPA-to-Votrax translation, and data transfer between the IBM-PC and HERO robot.

General Information

Program Structure

The HEROTALK system consists of three major modules: HEROTALK.COM, EDIT_IPA.CHN, and FINISHUP.CHN. HEROTALK.COM represents the major module of the system and serves as the controller during execution, processing the system's translation rules as well as the user's input of English text. HEROTALK.COM transfers program control to one of the other two modules contingent on choices made by the user during program execution. EDIT_IPA.CHN contains routines which allow the user to modify the program's IPA translation, whereas FINISHUP.CHN converts our IPA translation into Votrax phonemes and performs the data transfer between the IBM-PC and HERO robot, subsequently producing speech. Specific features in these modules

are described throughout this chapter, as well as a discussion of flow control between these modules.

Several other files comprise the HEROTALK system. Four files contain binary images, and represent "screens" which are displayed on the monitor at different points during program execution. By using stored binary images, as opposed to creating each display screen at run-time, execution speed is increased. These files have the extension "SCR" as part of their name. Several "help" files are also included as part of the HEROTALK system, and are accessed, upon user selection, at various points during program execution. Help files have the extension "HLP" as part of their name. Finally, four files are created with each text-to-speech translation performed by HEROTALK. These files save the various translation stages of our English text as generated by HEROTALK.

Data Structures Used

The primary data structure used throughout the HEROTALK system is the linked list. Both linked lists and doubly linked lists, implemented with pointers, are utilized heavily throughout the system. As the size of the English text sample entered by the user varies with each execution, the need for a dynamic data structure existed. Similarly, as the length of each rule string processed in the "letter-to-sound" rules employed varies, a similar storage feature was required. Furthermore, as was stressed in the previous chapter, bi-directional traversal of both these groups of information is necessary. Conse-

quently, doubly linked lists are used. In cases where bidirectional access is not essential, linked lists are employed. Also, in the module EDIT_IPA.CHN, the "array" is used in implementing the IPA editor.

Implementation

As a quick overview the HEROTALK system basically allows the user to enter into the system a sample of English text, and receive as output that text translated into speech phonemes. In this particular system those phonemes can be subsequently used with the HERO robot to enable the robot to speak the text entered by the user. Fig. 5-1 shows a simplified flowchart of the entire HEROTALK system. As is shown, upon invoking HEROTALK our "letter-to-sound" rules are read, processed, and stored in memory. Next, the user's English text is processed and also stored in memory. At this point, the user may also define new, temporary translation rules for the system. Upon processing of all user input the "text-to-IPA" translation is performed. Next, the original English text entered is displayed next to its IPA counterpart. If discrepancies exist, the user has the option of modifying the IPA translation via use of the "IPA Editor," a screen editor which allows editing of the IPA translation. Once the user is satisfied with the system's IPA translation, it is saved to disk, and subsequently translated into Votrax phonemes. At this point the translation process is complete though in this implementation, because of the data format requirements of the

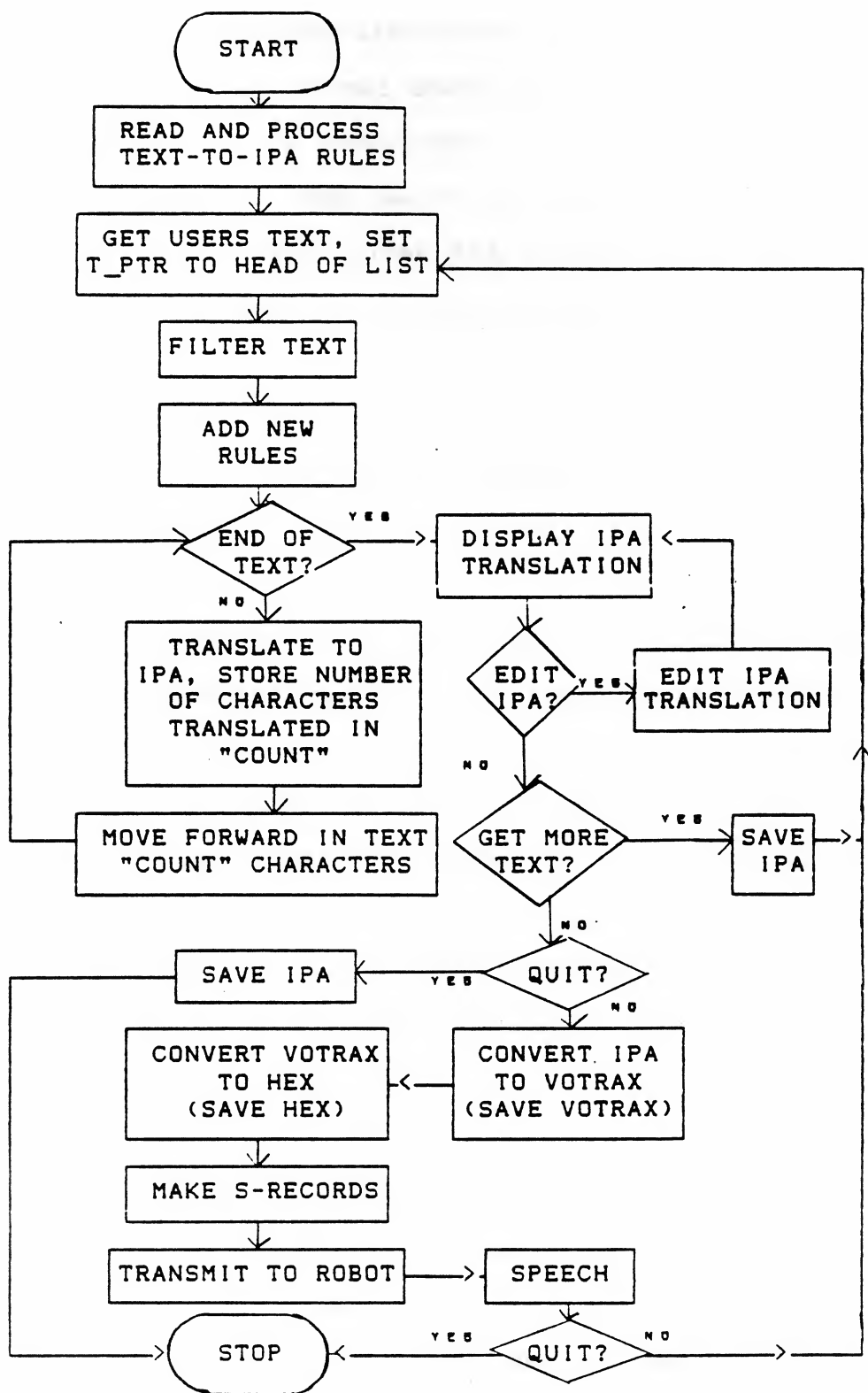


Fig. 5-1 - Flowchart of the HEROTALK System

HERO robot, the phonemes are formatted into "Motorola S-Records," a file transfer format enabling the successful transfer of our phonemes to the HERO robot, subsequently enabling the production of speech. The user's previously entered text is then spoken and the user given the option of continuing with additional text translation or exiting the program.

The HEROTALK Modules

Module HEROTALK.COM

As previously mentioned, HEROTALK.COM represents the main module of the HEROTALK system. The tasks most vital to the system are performed in it including the processing and loading into memory of the "letter-to-sound" rules used in the translation of English text-to-IPA phonemes as well as the processing and loading into memory of the user's English text. Also performed in this module is the translation of English text to IPA phonemes. Finally, it is from this module that flow control of the HEROTALK system is monitored.

The Text-to-IPA Translation Rules

The "letter-to-sound" rules used in HEROTALK are found in the external file RULES.DAT in the form of character strings. Twenty-eight such strings exist, one for each letter of the alphabet, accounting for the first twenty-six, and two additional strings: one for numeric characters, and one for

nonalphanumeric characters. The strings are arranged alphabetically and separated from one another by the character "|." As stated in Chapter IV, the rule set used by HEROTALK is based on the set developed by Elovitz, et. al., originally consisting of 329 letter-to-sound rules. The set used here is somewhat larger, consisting of 418 rules. Additions were made to handle abbreviations, acronyms and initials, as well as numerical digits and several nonalphanumeric characters. In addition, using statistics compiled by Kucera in his analysis of the English language [13], words of high frequency use were run through the translation system, and their IPA translation examined for accuracy. For those words which translated incorrectly, specific translation rules were added to ensure their correct translation.

Upon invoking HEROTALK, the file RULES.DAT is opened, and its contents read and stored in RAM with each particular rule string stored individually in the form of a doubly linked list. In the HEROTALK system, twenty-eight global pointer variables are defined -- one for each letter of the alphabet (a_rules-z_rules), one for the numeric characters (num_rules), and one for the nonalphanumeric characters (punc_rules). The algorithm employed in processing the "letter-to-sound" rules sequentially reads characters from RULES.DAT, one at a time, constructing with them a doubly linked list. When the character "|" is encountered, we know we have reached the end of the first rule string in the rule set, and the variable "a_rules" is assigned as the head of the constructed list. The process then repeats

with the next string constructed with "b_rules" as the head of the list. The process repeats twenty-eight times; once for each letter of the alphabet, once for the numeric characters, and finally once for the nonalphanumeric characters. This process results in the creation of twenty-eight separate doubly linked lists, one for each of the aforementioned character groups, in the PC's RAM. By storing each unique string of "letter-to-sound" rules as a separate doubly linked list the HEROTALK system, during text-to-IPA translation, can isolate a particular rule string immediately, thereby greatly speeding up the translation process. These doubly linked lists of rules exist in memory throughout the execution of HEROTALK. Consequently, should the user choose to continue at the end of one complete cycle of the translation process, the rules are not read and processed again.

User Input

Once our rules have been processed, we next need a sample of text to translate. Two methods exist by which the user may enter a text sample to HEROTALK; he may enter the text directly from the keyboard or he may have pre-prepared a datafile of text which can be read by the HEROTALK system. Input via the keyboard is, for the most part, self-explanatory. The user simply types in the text he wishes to have translated into speech phonemes. Input via a pre-prepared datafile is equally self-explanatory, with the user needing only to provide HEROTALK with the name of the file which contains the text the user

wishes to have translated. In either case the text entered is again stored in the form of a doubly linked list. As the length of the text sample varies with each execution of HEROTALK such a data structure provides the most flexibility while maintaining efficient use of memory.

In handling the user's text input, two problems can arise. First, the possibility exists that the user will enter, as part of his text, control characters, etc., which cannot be translated. Second, there exists the possibility that the user will provide HEROTALK with gibberish as his English text. As a solution to the former problem a filtering routine has been provided which filters out all untranslatable characters from our list of English text prior to its submittance to the translation routines. The filtering algorithm employed constructs a set of "bad" characters which should be removed. Each character of our text is compared to this set, with common characters deleted from our list of text. The characters filtered out include all control characters, as well as: "\", " ^, " _, " |, " and [DEL]. With regards to the latter problem, the user must be trusted to enter a legitimate sample of text. Because of the default rules included in the letter-to-sound rules employed, HEROTALK will generate a translation regardless of the text entered. Unless, however, the text upon which the translation is based is legitimate, the translation will be worthless.

One final means of user input concerns the feature which allows the user, while executing HEROTALK, to define new, but

temporary translation rules. An obvious problem when attempting to develop an unlimited vocabulary "text-to-speech" translator is devising a means of dealing with all possible cases which can arise in the translation process. The "letter-to-sound" rules used in HEROTALK attempt to translate any word given them by the user. As can be expected, however, the vast number of exceptions in the English language make it virtually impossible to construct a set of rules possessing this capability. To even attempt to do so, such a set would have to be extremely large. As HEROTALK runs on a PC with, in most cases, limited memory resources, a large set of rules is neither desirable, nor in some implementations possible. HEROTALK instead allows the user to define new, temporary rules during program execution. This, in effect, makes the rule set expandable during any given execution, yet keeps the original rule set intact and unmodified, therefore keeping the original set small and memory efficient. Of course, the rule set temporarily grows with each new addition, but normally the number of additions will not be so numerous as to cause memory problems. In the event that memory usage approaches exhaustion, the HEROTALK system prevents further rule addition until additional memory becomes available.

To add a new rule, the user must enter both the word needing the new rule, in English text, and its corresponding IPA translation. These two parts are then constructed in the "A[B]C=/D/" format (as discussed in the previous chapter), with the B string corresponding to the English text entered,

the A and C strings being blanks (to ensure that only B strings which match exactly to the English text entered for this new rule are translated with it), and the D string corresponding to the IPA translation entered. New rules are added to the rule string corresponding to the first letter of a new rule's B string, and are inserted, into this list, as the first rule. This ensures that new rules will "fire" before all other rules in the string, thereby enabling our new rule to override all other rules in the same string. New rules are added only to the rule lists stored in RAM, and not to the rule strings in RULES.DAT. Consequently, we need not worry about the user corrupting the system's permanent rule set. Also, note that a new rule will not affect the applicability of the original rules of the rule set. Because we have surrounded the English text in the new rule with blanks (i.e., both the A and C strings in the new rule consist of a blank), only parts of the text which match the B string exactly, and are isolated (i.e., not a part of another word) will match with this rule.

The addition of new rules to the set is usually done prior to the entering of the English text to be translated or between consecutive runs of the HEROTALK system during the same initial execution. New additions exist as long as a given execution of HEROTALK, or in other words, until the HEROTALK system is exited and the user returns to the DOS level. Finally, keep in mind that these additions are only temporary. Should the user wish to make permanent additions to the rule set, he may do so by editing the file RULES.DAT.

Text-to-IPA Translation

At this point, the HEROTALK system consists of twenty-eight unique strings of letter-to-sound rules, stored in RAM as separate doubly linked lists, and a doubly linked list of English text which is to be translated into IPA phonemes. The next step is the actual translation of text-to-IPA. As this was discussed in-depth in the preceding chapter, it will be omitted here. Once the IPA translation is complete, program control is transferred to module "EDIT_IPA.CHN."

Module EDIT_IPA.CHN

This module's major function is to enhance the IPA translation. As the major problem throughout the design of HEROTALK is the generation of a "flawless" IPA translation, EDIT_IPA.CHN, as its name implies, allows the user to modify the IPA translation and improve its accuracy.

After control is passed from HEROTALK.COM, the display monitor is divided into three sections. The top half becomes an "IPA Key," displaying the IPA set of phonemes along with sample words highlighting the sound associated with each phoneme. The lower half is similarly cut in half with the left half used to display the English text entered by the user and the right half used to display that text's IPA translation. Using all three windows, the user is given the opportunity to check the IPA translation computed by the HEROTALK system. This "check" opportunity serves two purposes: first, it allows the user to determine whether the IPA translation needs edit-

ing, and second, it can be used as a guide for identifying words which should be added as new, temporary rules in subsequent translations, or possibly permanent modifications to the HEROTALK rule set.

After the entire sample of English text, and its corresponding IPA translation are displayed, a menu of options appears allowing the user to either continue with the text-to-speech process, quit and return to DOS level, or edit the IPA translation.

The IPA Editor

The IPA editor is a screen editor which allows the user to make modifications to HEROTALK's IPA translation. The editor is based on an 80 x 75 array, corresponding to an editing buffer of 75 lines each 80 characters in length. An array is used because it creates an easy analogy between the display on the screen, and the contents of the editing buffer. The size restriction of the buffer could perhaps cause problems if the IPA list exceeds the size of the array allotted. It is highly unlikely however, as such an IPA list would be derivative of a sample of English text which would be too large to remain either practical or realistic. In the event such a list is produced by the system, HEROTALK still runs to completion; only use of the editor is disabled.

Upon invoking the editor, the complete IPA translation is loaded into the editor buffer. Consequently, any modifications made to the IPA translation are made to a copy of the transla-

tion, and not the original. After the IPA translation has been loaded the lower half of the display screen becomes a screen editor and the IPA translation of the user's text, or fraction of it (the window of the editor permits only 7 lines of the buffer to be displayed at a time), is displayed ready for editing. In the upper half of the display screen, the IPA key remains as an aid during editing. In most cases, any necessary editing will usually entail merely changing a phonetic sound to enhance the translation.

Use of the editor is, for the most part, self-explanatory though two features merit attention. The first allows the user to edit a line on the screen and then restore it if necessary to its original state. As the IPA translation is paramount in completing the remainder of the text-to-speech process, it is imperative that it not be corrupted by the editor. Consequently, if the user edits a line and edits it into disarray the line can be restored. Unfortunately, however, a line may be restored only so long as the user does not reposition the cursor, while using the editor, to a different line of IPA text. This restriction brings us to the second important feature -- a means of restoring the entire IPA translation back to its original state. Such a feature allows the user to edit several lines of the translation, inadvertently destroy it, and still be able to restore it to its original condition. Both features make using the IPA editor less prone to upsetting the text-to-speech process. Nonetheless, the user must exercise caution.

Once editing is complete, the user is again offered the menu of options as offered when he entered the editor. At this point, as before, the user can quit, continue with the text-to-speech translation, enter additional English text prior to continuing with the text-to-speech translation, or, of course, edit the translation again. The choice made greatly affects the flow of control. If the user chooses to enter additional text, control is passed back to HEROTALK.COM and the user input routines repeat. If the user chooses to continue with the text-to-speech process, control is passed to module FINISHUP.CHN. In both of these cases the IPA translation is saved to disk, and then deleted from memory freeing up resources for other parts of the system. If the user chooses to quit, the IPA translation is saved and the user returned to the DOS level.

Module FINISHUP.CHN

Upon transfer of control to FINISHUP.CHN roughly half of the two-step translation process is complete. In fact the text-to-speech process in some respects is complete as we have successfully converted a sample of English text to a standard phonetic representation. It is in this module where the HEROTALK system can be tailored to be used with any phonetic based speech synthesis unit. With that in mind, the major purpose of this module is the conversion of IPA phonemes to the phoneme set particular to the speech synthesizer to be used. The speech synthesizer used in this implementation is a Votrax

SC-01, and thus in this module our IPA translation is converted to Votrax phonemes.

IPA-to-Votrax Translation

The translation of IPA-to-Votrax phonemes is accomplished by associating with each IPA phoneme an equivalent Votrax phoneme or combination of phonemes. Table 5-1 shows the Votrax phoneme set [22], while Table 5-2 shows the translation correspondence used by HEROTALK to convert our IPA translation to its equivalent Votrax representation. As can be seen not all 64 phonemes of the Votrax set are used; only those sounds which produced the most intelligible speech from the IPA phonetic transcription are used. Basically, the correspondence was developed through repeated experimentation with the Votrax set. For the most part the consonants, and consonant sounds, were not difficult to translate with most IPA sounds having an equivalent sound in the Votrax set. The two major exceptions were the combination of the "T" and "CH" Votrax phonemes to produce the IPA "CH" sound, and the combination of the "D" and "J" Votrax phonemes to produce the "JH" sound in IPA. A more formidable obstacle was arriving at Votrax combinations to adequately reproduce the IPA "vowel" sounds (basically, IY, IH, EY, EH, AE, AA, AO, OW, UH, UW, AX, AH, AY, AW, and OY). In generating this correspondence, a series of words highlighting each voiced phoneme was created and run through the HEROTALK system to generate their respective IPA translations. Next, using the HERO robot's "Voice Dictionary" [26], a combination

Sym.	Dur.	Ex.	Sym.	Dur.	Ex.	Sym.	Dur.	Ex.
A	185	day	ER	146	bird	S	90	pass
A1	103	made	F	103	fast	SH	121	shop
A2	71	made	G	71	get	T	71	tap
AE	185	dad	H	71	high	TH	71	thin
AE1	103	after	I	185	pin	THV	80	the
AH	250	mop	I1	121	in	U	185	move
AH1	146	body	I2	80	bit	U1	90	you
AH2	71	on	I3	55	bit	UH	185	cup
AW	250	call	IU	59	you	UH1	103	uncle
AW1	146	law	J	47	jar	UH2	71	about
AW2	90	salt	K	80	car	UH3	47	run
AY	65	day	L	103	land	V	71	van
B	71	bag	M	103	mat	W	80	win
CH	71	chop	N	80	sun	Y	103	any
D	55	fade	NG	121	ring	Y1	80	yard
DT	47	but	O	185	cold	Z	71	zoo
E	185	meet	O1	121	board	ZH	90	azure
E1	121	be	O2	80	for	PAO	47	*
EH	185	get	OO	185	book	PA1	185	*
EH1	121	head	OO1	103	look	STOP	47	*
EH2	71	end	P	103	past	(* - no sound)		
EH3	59	omlet	R	90	red			

Sym: Phoneme Symbol Dur: Duration in ms Ex: Sample Word

Table 5-1 - The Votrax Phoneme Set

IPA	Votrax	IPA	Votrax
AA	- AH1 UH3	M	- M
AE	- AE1 EH3	N	- N
AH	- UH1 UH2	NX	- NG
AO	- AW	OW	- O1 U1
AW	- AH1 UH3 W	OY	- O1 UH3 I3 AY
AX	- UH1	P	- P
AY	- AH1 EH3 Y	R	- R
B	- B	S	- S
CH	- T CH	SH	- SH
D	- D	T	- T
DH	- THV	TH	- TH
EH	- EH1 EH3	UH	- O01
ER	- ER	UW	- U1 U1
EY	- A1 AY Y	V	- V
F	- F	W	- W
G	- G	WH	- W
HH	- H	Y	- Y1
IH	- I3 I2	Z	- Z
IY	- E1 Y	ZH	- ZH
JH	- D J	\$	- PA1
K	- K	*	- PA1 PA1
L	- L	~	- PAO

Table 5-2 - The IPA-to-Votrax Phoneme Correspondence

of Votrax phonemes was constructed for each corresponding IPA "vowel." The aforementioned words were then generated with the Votrax SC-01 using these Votrax combinations. In some cases the combinations produced intelligible speech and those combinations were not altered. In those cases where the speech was not intelligible, the Votrax phoneme combination was modified and the words sounded again. The modifications usually involved increasing or decreasing the duration of a particular voiced sound. When the series of words was spoken clearly and intelligible enough to be clearly understood, the Votrax equivalent for that IPA sound was accepted as part of the translation set and no longer modified.

The translation of each IPA phoneme to its equivalent Votrax combination is performed via a "case" statement. Each IPA phoneme is isolated from the IPA translation, located in the "constant" part of this "case" statement, and its "statement" part, corresponding to its equivalent Votrax combination, copied to a list of Votrax phonemes. This "case" statement, included in the procedure "IPA_TO_HERO" in the module, represents the major modification necessary to the HEROTALK system which allows the system to function with virtually any phonetic based speech synthesizer. All that's needed to be done, in fact, is to compute for each IPA phoneme its equivalent combination in the phoneme set of the speech synthesizer to be used, and then replace each IPA phoneme's equivalent, in the code, with that combination.

At this point for most speech synthesizers the phonemes

generated would now be sent to it and speech would be generated. In this implementation however, because we are using the Votrax SC-01 built into the HERO robot, the Votrax phonemes generated must first be converted into their equivalent "HERO robot" hexadecimal representation and then formatted into "data records" which enable transfer between the PC and the HERO robot. As this would not be required by most other speech synthesizers which could be used with HEROTALK, the remaining steps in the "text-to-speech" process would probably be unnecessary if another speech unit were used.

In order to send our Votrax phonemes to the HERO robot, they must be formatted into "Motorola S-Records," a data transfer format allowing data transfer between, in our case, the IBM-PC and the HERO robot. Each data record consists of a string of characters comprising five separate fields: a "type" field, a "record length" field, an "address" field, a field for code and/or data, and a "checksum" field. More about the "S-Record" format is given in the description of the "S-Record" format in Appendix B.

Associated with each Votrax phoneme is a two-character hexadecimal code [26]. These codes are substituted for the Votrax phonemes in our "S-Records," comprising the data fields in these records. Consequently, one additional translation takes place in the FINISHUP.CHN module -- the conversion of Votrax phonemes to their respective hexadecimal codes. This is accomplished, as was our IPA-to-Votrax translation, with a "case" statement. Each Votrax phoneme is isolated from the

Votrax translation, "looked up" in the "case" statement, and its corresponding hexadecimal code added to a list of hexadecimal phonemes. After all Votrax phonemes are converted to their hexadecimal codes, the hexadecimal list is then formatted into "S-Records" and subsequently ready for transfer to the HERO robot.

Fig. 5-2 summarizes the text-to-speech translation process and shows the various translation stages generated by the HEROTALK system. Shown is the progression from English text sample to IPA equivalent, Votrax equivalent, hexadecimal equivalent, and finally formatted "S-Records."

Upload

The final step then in this implementation is the transfer of our Votrax phonemes (in hex form, and formatted as Motorola S-Records) to the HERO robot. The transfer is done with communications parameters of 9600 baud, no parity, 8 data bits, and 1 stop bit, and using the PC's COM1 port. These parameters can be modified in procedure "SetUp_UART." To enable the HERO robot to speak our text, a small speech program [27] must be loaded into the robot, instructing it to speak the phonemes generated by HEROTALK. The HERO robot has 3750 bytes of user available memory (i.e., memory available for the loading of programs, and/or speech phonemes). Our speech program consumes only 5 bytes of memory, thereby leaving 3745 bytes of memory available and thus the potential of speaking 3745 successive phonemes (each phoneme consumes only 1 byte of memory). How

For the body of a paper, margins should be approximately one inch at each edge. Between text and footnotes there should be a space about three-quarters of an inch, divided midway by a short line from the left margin. In typing, a double space with a line made by the underscore key and then another double space will provide a proper division between text and footnotes.

Fig. 5-2(a) - A Sample Of English Text for the HEROTALK System

F AO R ~/DH AX ~/B AA D IY ~/AX V ~/AX ~/P EY P ER \$ /M AA R JH
 IH N Z ~/SH UH D ~/B IY ~/AE P R AA K ~ S IH M AE T L IY ~/W AH
 N ~/IH N CH ~/AE T ~/IY CH ~/EH D JH * /B EH T W IY N ~/T EH K
 ~ S T ~/AE N D ~/F UW T N OW T S ~/DH EH R ~/SH UH D ~/B IY
 ~/AX ~/S P EY S ~/AE B AW T ~/TH R IY K W AO R T ER Z ~/AX V
 ~/AE N ~/IH N CH \$ /D IH V AY D IH D ~/M IH D W EY ~/B AY ~/AX
 ~/SH AO R T ~/L AY N ~/F R AA M ~/DH AX ~/L EH F T ~/M AA R JH
 IH N * /IH N ~/T AY P IH NX \$ /AX ~/D AW B L ~/S P EY S ~/W IH
 TH ~/AX ~/L AY N ~/M EY D ~/B AY ~/DH AX ~/AH N D ER S K AO R
 ~/K IY ~/AE N D ~/DH EH N ~/AE N AH DH ER ~/D AW B L ~/S P EY S
 ~/W IH L ~/P R AH V AY D ~/AX ~/P R OW P ER ~/D IH V IH ZH AX N
 ~/B EH T W IY N ~/T EH K ~ S T ~/AE N D ~/F UW T N OW T S * /

Fig. 5-2(b) - The "unedited" IPA Translation of the English Text Sample in Fig. 5-2(a)

F AO R ~/DH AX ~/B AA D IY ~/AX V ~/AX ~/P EY P ER \$ /M AA R JH
 IH N Z ~/SH UH D ~/B IY ~/AX P R AA K ~ S IH M AX T L IY ~/W AH
 N ~/IH N CH ~/AE T ~/IY CH ~/EH D JH * /B EH T W IY N ~/T EH K
 ~ S T ~/AE N D ~/F UH T N OW T S ~/DH EH R ~/SH UH D ~/B IY
 ~/AX ~/S P EY S ~/AX B AW T ~/TH R IY K W AO R T ER Z ~/AX V
 ~/AE N ~/IH N CH \$ /D IH V AY D IH D ~/M IH D W EY ~/B AY ~/AX
 ~/SH AO R T ~/L AY N ~/F R AA M ~/DH AX ~/L EH F T ~/M AA R JH
 IH N * /IH N ~/T AY P IH NX \$ /AX ~/D AH B AX L ~/S P EY S ~/W
 IH TH ~/AX ~/L AY N ~/M EY D ~/B AY ~/DH AX ~/AH N D ER S K AO
 R ~/K IY ~/AE N D ~/DH EH N ~/AE N AH DH ER ~/D AH B AX L ~/S P
 EY S ~/W IH L ~/P R OW V AY D ~/AX ~/P R AA P ER ~/D IH V IH ZH
 AX N ~/B EH T W IY N ~/T EH K ~ S T ~/AE N D ~/F UH T N OW T S
 * /

Fig. 5-2(c) - The "edited" IPA Translation of the English Text Sample in Fig. 5-2(a)
 (modifications made with the IPA Editor are underlined)

F AW R PA0 THV UH1 PA0 B AH1 UH3 D E1 Y PA0 UH1 V PA0 UH1 PA0
P A1 AY Y P ER PA1 M AH1 UH3 R D J I3 I2 N Z PA0 SH OOl D PA0 B
E1 Y PA0 UH1 P R AH1 UH3 K PA0 S I3 I2 M UH1 T L E1 Y PA0 W UH1
UH2 N PA0 I3 I2 N T CH PA0 AE1 EH3 T PA0 E1 Y T CH PA0 EH1 EH3
D D J PA1 PA1 B EH1 EH3 T W E1 Y N PA0 T EH1 EH3 K PA0 S T PA0
AE1 EH3 N D PA0 F OOl T N O1 U1 T S PA0 THV EH1 EH3 R PA0 SH
OOl D PA0 B E1 Y PA0 UH1 PA0 S P A1 AY Y S PA0 UH1 B AH1 UH3 W
T PA0 TH R E1 Y K W AW R T ER Z PA0 UH1 V PA0 AE1 EH3 N PA0 I3
I2 N T CH PA1 D I3 I2 V AH1 EH3 Y D I3 I2 D PA0 M I3 I2 D W A1
AY Y PA0 B AH1 EH3 Y PA0 UH1 PA0 SH AW R T PA0 L AH1 EH3 Y N
PA0 F R AH1 UH3 M PA0 THV UH1 PA0 L EH1 EH3 F T PA0 M AH1 UH3 R
D J I3 I2 N PA1 PA1 I3 I2 N PA0 T AH1 EH3 Y P I3 I2 NG PA1 UH1
PA0 D UH1 UH2 B UH1 L PA0 S P A1 AY Y S PA0 W I3 I2 TH PA0 UH1
PA0 L AH1 EH3 Y N PA0 M A1 AY Y D PA0 B AH1 EH3 Y PA0 THV UH1
PA0 UH1 UH2 N D ER S K AW R PA0 K E1 Y PA0 AE1 EH3 N D PA0 THV
EH1 EH3 N PA0 AE1 EH3 N UH1 UH2 THV ER PA0 D UH1 UH2 B UH1 L
PA0 S P A1 AY Y S PA0 W I3 I2 L PA0 P R O1 U1 V AH1 EH3 Y D PA0
UH1 PA0 P R AH1 UH3 P ER PA0 D I3 I2 V I3 I2 ZH UH1 N PA0 B EH1
EH3 T W E1 Y N PA0 T EH1 EH3 K PA0 S T PA0 AE1 EH3 N D PA0 F
OOl T N O1 U1 T S PA1 PA1

Fig. 5-2(d) - The Votrax Equivalent of the Edited IPA
Translation as Computed in Fig. 5-2(c)

1D 3D 2B 03 38 32 03 0E 15 23 1E 3C 22 03 32 0F 03 32 03 03 06
21 29 25 3A 3E 0C 15 23 2B 1E 1A 09 0A 0D 12 03 11 16 1E 03 0E
3C 22 03 32 25 2B 15 23 19 03 1F 09 0A 0C 32 39 18 3C 22 03 2D
32 31 0D 03 09 0A 0D 2A 10 03 2F 00 39 03 3C 22 2A 10 03 02 00
1E 1E 1A 3E 3E 0E 02 00 39 2D 3C 22 0D 03 2A 02 00 19 03 1F 2A
03 2F 00 0D 1E 03 1D 16 2A 0D 35 37 2A 1F 03 38 02 00 2B 03 11
16 1E 03 0E 3C 22 03 32 03 1F 03 06 21 29 1F 03 32 0E 15 23 2D
39 03 39 2B 3C 22 19 2D 3D 2B 2A 3A 12 03 32 0F 03 2F 00 0D 03
09 0A 0D 2A 10 3E 1E 09 0A 0F 15 00 29 1E 09 0A 1E 03 0C 09 0A
1E 2D 06 21 29 03 0E 15 00 29 03 32 03 11 3D 2B 2A 03 18 15 00
29 0D 03 1D 2B 15 23 0C 03 38 32 03 18 02 00 1D 39 03 0C 15 23
2B 1E 1A 09 0A 0D 3E 3E 09 0A 0D 03 2A 15 00 29 25 09 0A 14 3E
32 03 1E 32 31 0E 32 18 03 1F 03 06 21 29 1F 03 2D 09 0A 39 03
32 03 18 15 00 29 0D 03 0C 06 21 29 1E 03 0E 15 00 29 03 38 32
03 32 31 0D 1E 3A 1F 19 3D 2B 03 19 3C 22 03 2F 00 0D 1E 03 38
02 00 0D 03 2F 00 0D 32 31 38 3A 03 1E 32 31 0E 32 18 03 1F 03
06 21 29 1F 03 2D 09 0A 18 03 03 2B 35 37 0F 15 00 29 1E 03 32
03 03 2B 15 23 25 3A 03 1E 09 0A 0F 09 0A 07 32 0D 03 0E 02 00
39 2D 3C 22 0D 03 2A 02 00 19 03 1F 2A 03 2F 00 0D 1E 03 1D 16
2A 0D 35 37 2A 1F 3E 3E FF

Fig. 5-2(e) - The HEX Equivalent of Fig. 5-2(d)

S00600004844521B
S108004072004520FEE2
S11700451D3D2B033832030E15231E3C2203320F033203036D
S1170059062129253A3E0C15232B1E1A090A0D120311161E81
S117006D030E3C220332252B152319031F090A0C3239183C36
S117008122032D32310D03090A0D2A10032F0039033C222A52
S1170095100302001E1E1A3E3E0E0200392D3C220D032A025C
S11700A90019031F2A032F000D1E031D162A0D35372A1F0358
S11700BD3802002B0311161E030E3C220332031F0306212965
S11700D11F03320E15232D3903392B3C22192D3D2B2A3A122E
S11700E503320F032F000D03090A0D2A103E1E090A0F150090
S11700F9291E090A1E030C090A1E2D062129030E1500290368
S117010D3203113D2B2A03181500290D031D2B15230C0338D2
S117012132031802001D39030C15232B1E1A090A0D3E3E09D2
S11701350A0D032A15002925090A143E32031E32310E321898
S1170149031F030621291F032D090A39033203181500290DF3
S117015D030C0621291E030E1500290338320332310D1E3A86
S11701711F193D2B03193C22032F000D1E033802000D032F83
S1170185000D3231383A031E32310E3218031F030621291F10
S1170199032D090A1803032B35370F1500291E033203032B85
S11701AD1523253A031E090A0F090A07320D030E0200392D8E
S11701C13C220D032A020019031F2A032F000D1E031D162A6A
S10B01D50D35372A1F3E3EFFE1
S9030000FC

Fig. 5-2(f) - The S-Records which contain the
HEX translation shown in Fig. 5-2(e)

this translates into the number of words potentially speakable varies depending on the number of phonemes required to speak a given word; larger words, of course, require more phonemes and thus more memory. At any rate, the memory available provides more than enough room to speak more than a page of text. After the phonemes are loaded the speech program can be executed on the robot, and the user's English text spoken.

After our "S-Records" have been uploaded to the HERO robot, the user has the option of quitting, or entering an additional sample of English text, subsequently generating more speech phonemes and thus more speech. If the user chooses to enter a new sample, program control is returned to module HEROTALK.COM and the entire translation process repeats. The major difference in repeated translations is the absence of the "letter-to-sound" rule processing which occurs upon invoking HEROTALK. As the translation rules are never deleted from memory this process need not be repeated on successive translations during the same initial execution. Also keep in mind that any "new" translation rules entered since invoking HEROTALK remain in effect.

For each complete English text-to-speech translation, four data files are created. These files correspond to the IPA and Votrax translations of the user's English text, the hexadecimal equivalent of the Votrax translation (i.e., the hexadecimal codes representing the phonemes generated by the HEROTALK system), and finally the numerous "S-Records" generated by the system. These files are saved to disk as a record for the user

of the various translations computed by the system. The IPA file can be used as a guideline for manually translating text to speech phonemes for another phoneme-based speech synthesizer. Also, the hexadecimal translation can be used to provide the hexadecimal codes required for speech and incorporated into other programs written for the HERO robot thereby allowing a user of the robot to incorporate unlimited vocabulary into his robot programs by merely incorporating this "hex" translation into his program. Finally, the "S-Records" are saved so that the user may preserve any saying for later execution on the robot. Using any file transfer program the "S-Records" can be uploaded to the robot, and the speech embodied in the records spoken on command. The Votrax set is saved primarily as a record for the user.

Chapter VI

Results And Discussion

In this chapter HEROTALK's strengths and limitations are addressed, and suggestions for improvements to the system are proposed. In addition, results of testing the system are discussed, and the HEROTALK system is compared to two commercial text-to-speech systems: the IBM-PCjr Speech Attachment and the Macintosh "Smooth Talker" speech system.

Strengths Of The HEROTALK System

The most prevalent strength of the HEROTALK system is its unlimited vocabulary capability. The fact that it can translate almost any English word given it as input certainly gives it an advantage over text-to-speech systems which lack this capability. Equally advantageous is its two-stage translation process. By converting English text first to IPA phonemes, and then from IPA phonemes to Votrax phonemes, the system can be easily modified to function with virtually any phoneme-based speech synthesizer by merely replacing its Votrax conversion set with phonemes applicable to the speech synthesizer to be used. The modular design of HEROTALK aids in this applicability. As the text-to-IPA and IPA-to-Votrax routines are embodied in different modules of the system, replacing the Votrax set with another phoneme set requires only a minimal under-

standing of the system's structure, and a subsequent recompilation of only one of the three modules which comprise it. Furthermore, a module can be easily modified or expanded provided that its functions and resultant output remain consistent with that expected by the other modules of the system.

Finally, the letter-to-sound rules used by HEROTALK are modifiable by the user. Consequently, inadequacies in the rule set can be easily modified and even new rules added as the user deems necessary.

Limitations Of The HEROTALK System

HEROTALK's most obvious limitation is its "batch" operating mode. This is more a restriction enforced by its design specifications, however, rather than an inherent limitation of the system itself. The HERO robot does not facilitate the sending of small groups of phonemes to its Votrax speech synthesizer. For this reason, a user's entire translated text is sent to the speech synthesizer at once, rather than in parts as is the case with "interactive" text-to-speech systems. As an "interactive" mode would allow the user more control over speech production, its absence here certainly hinders HEROTALK's applicability in a real-time speech setting. The program could be easily modified, however, to overcome this limitation.

An additional weakness of the system is the time it consumes in generating its speech translation. This delay could be reduced however with some minor system modifications. As

HEROTALK was designed as a speech synthesis tool to be used in conjunction with the HERO educational robot, it adheres to its educational premise. Consequently, as HEROTALK executes, the user is constantly informed of the progress of its text-to-speech translation. This monitoring includes periodic screen displays throughout the process informing the user of the system's current stage of translation. By modifying the source code and removing these informative but nonessential delays the text-to-speech translation time could be noticeably improved.

Suggestions For Improvement

Obviously, an immediate improvement to the system would be to modify the code, and enable the system to run in an interactive mode. A more ambitious enhancement to the system would be to incorporate a means of adding syllable stress and pitch inflection to the phonetic translation. As the system currently stands the phonetic translation produced contains no syntactical enhancement, and thus produces no intonation in its subsequently synthesized speech. By incorporating this enhancement into the translation, the quality of the speech produced from the system would be much closer to that of human speech. Syntactical enhancement could be incorporated into the already existing "letter-to-sound" rules, or more likely included as a "post-processor" of the text-to-IPA translation, and added after the text is translated into IPA phonemes. Algorithms have been developed which perform this task [7, 24], and the interested reader is referred to these for additional

information.

An additional suggestion for improvement would be to incorporate some means of modifying, from within the program, the phoneme set which IPA phonemes are translated to. As the system now exists, to change from the Votrax set to another set, and thus another speech synthesizer, the source code of module FINISHUP.CHN must be externally modified, and the Votrax phonemes in that source replaced with phoneme codes corresponding to the speech synthesizer to be used. After these modifications the module must then be recompiled. Whereas modifying the source code of the system requires a knowledge of programming and a familiarity with Turbo Pascal, the aforementioned improvement to the system would not, thereby enabling nearly any user to easily adapt the system to another speech synthesis unit. This enhancement could be added as an interactive menu-driven "phoneme-by-phoneme" modification process, or entire phoneme sets corresponding to different speech synthesizers could be saved to disk, and read into the system upon user direction at run time. With this improvement, not only would switching from one speech synthesizer to another be facilitated, thereby improving upon the applicability of the HEROTALK system, but also user modification of the source code avoided, and thus the risk of inadvertent corruption of the code avoided as well.

A final suggestion for improvement would be to supplement the system's letter-to-sound rules with a look-up dictionary to handle translation exceptions. A look-up dictionary was con-

sidered in the system's original design specifications, but was replaced with an analogous feature which allows the user to add the equivalent of a "look-up" dictionary, temporarily, to the rule set during program execution. A look-up dictionary would, on the other hand, provide permanent exception handling to the system. It's absence in the final design stage was prompted primarily as a means of avoiding not only the added time consumption inherent with searching a long list of exceptions, but to also avoid the memory consumption such a list would entail. In order to effectively use a list of exceptions, it too would have to be loaded into the PC's RAM, and not all PC's on which HEROTALK may be executed are configured with enough memory to store both the letter-to-sound rules needed by the system, and a look-up dictionary of word exceptions. On the other hand, a dictionary could be developed for a particular application and particular target machine relatively easy if the dictionary were designed around that machine's available resources.

Comparison of HEROTALK with the IBM-PCjr. Speech Attachment

The PCjr Speech Attachment, like HEROTALK, produces speech from English text. Both are microcomputer-based systems, and both incorporate no intentional prosody in their respective translations. The major difference between the two systems lies in the method of speech synthesis each employs; the PCjr system employs the waveform-encoded method of speech synthesis, whereas HEROTALK employs phoneme-generated speech. Consequent-

ly, as might be expected, the quality of the speech generated by the PCjr system is superior to the "robotic" sounding speech produced by HEROTALK. As might also be expected, however, the PCjr system is limited to a 196-word vocabulary, whereas HEROTALK has an unlimited vocabulary.

In using each system, HEROTALK runs as a separate program, whereas the PCjr system must be driven by codes embedded in the source code of BASIC programs. With regards to user input required by each system, all the user of HEROTALK must do is type English words when prompted by the system. To use the PCjr system on the other hand, the user must first identify the words to be spoken, identify their corresponding numeric codes from a table in the system's user manual, and then incorporate these codes into a BASIC program instructing the PCjr speech synthesizer to speak the words associated with these codes. Consequently, whereas a novice would have no problem using the HEROTALK system, the PCjr system requires at least an elementary knowledge of programming as well as a familiarity with the BASIC language.

The PCjr system does have the advantage of running interactively. Consequently, codes can be sent to the speech unit at the user's discretion and the words sounded almost immediately. HEROTALK could be used in similar fashion but because the user must manually execute speech programs on the HERO robot in order to use its Votrax speech unit, it would be impractical to send individual words to the unit and then manually execute the generated speech program separately for

each individual word. A final point to be noted is the fact that HEROTALK can be adapted to function with virtually any phoneme-based speech synthesizer. The PCjr system, on the other hand, can be used only with the IBM-PCjr.

Comparison of HEROTALK with the "Smooth Talker" System

The "Smooth Talker" System is a microcomputer-based text-to-speech system which runs on the Macintosh microcomputer. "Smooth Talker" is a phoneme-based speech system, and like HEROTALK uses pronunciation rules to convert text to speech. Whereas the HEROTALK system uses 418 letter-to-sound rules, "Smooth Talker" uses only 112. Consequently, not nearly as much memory is consumed by the translation rules in the "Smooth Talker" system. Furthermore, while HEROTALK's "letter-to-sound" rules are readable and modifiable by the user, "Smooth Talker"'s rules are not only unmodifiable by the user, but also unavailable for perusal. As "Smooth Talker" is a copyrighted commercial system, however, this is understandable. Like HEROTALK, "Smooth Talker"'s required input from the user consists merely of typed English text. This text, as with HEROTALK, can be entered either interactively via the keyboard, or passively via a pre-prepared data file. In both cases, the speech produced has the typical "robotic" sound, though "Smooth Talker"'s quality is enhanced somewhat by prosodic features incorporated into its translation.

Like HEROTALK, "Smooth Talker" provides a means of working with the translation at the phonetic level. Invoked with an

escape sequence during text input, the user can enter a "phonetic" mode which allows typing of words in their phonetic spelling. Consequently, words which the user feels will translate incorrectly can be phonetically spelled out in the input text, thereby improving the eventual speech translation. A similar feature exists in HEROTALK by invoking the IPA editor. In fact, use of HEROTALK's editor is somewhat easier in that while editing the IPA translation, a key displaying all IPA phonemes is displayed on the screen, directly above the IPA translation. Consequently, the user need not consult external tables, or the user's manual, to search for phoneme codes as is the case with "Smooth Talker." Finally, "Smooth Talker" is remarkably faster than HEROTALK in generating its synthesized speech. As discussed earlier, HEROTALK's translation speed could be improved with some simple modifications to the system. The translation speed of "Smooth Talker" would be difficult to match, however, as much of this difference in speed can be attributed to the slow operating speed of the IBM-PC in comparison to the Macintosh system.

Testing the HEROTALK System

The HEROTALK system generates a very accurate phonetic translation of English text. In an average text sample, the rules will generate correct phonetic pronunciations for between 85% and 90% of the words. In the example given in Chapter V (i.e., Fig. 5-2), HEROTALK translated correct phonetic spellings for 60 of the 67 words in the sample, or roughly 89%. In

the 7 incorrect spellings, as is the case with nearly all incorrect phonetic spellings generated by HEROTALK, the "errors" consist of a single erroneous vowel sound in each word and most of these single errors are easily correctable by the listener during speech production. Of course, much of this success rate depends on the type of words input to the system. Using HEROTALK to translate the first several hundred entries from a list of words ranked in the order of their frequency of occurrence in everyday English speech [13], HEROTALK correctly translated about 85% of them. For those words incorrectly translated, most were again the result of a single erroneous vowel sound. If, on the other hand, HEROTALK were given a list of scientific terms and/or proper names, obviously the success rate would drop. Of course, by using the IPA editor, or entering new, temporary rules for "problem words" during execution, HEROTALK can generate correct pronunciations for 100% of the words input to it.

Bibilography

1. Barbour, Andrew. "Computerized Speech: Talking Its Way into the Classroom," Electronic Learning, January 1987.
2. Bristow, Geoff. Electronic Speech Synthesis, McGraw Hill Books, 1984.
3. Cater, John P. Electronically Speaking: Computer Speech Generation, Howard W. Sams & Co., Inc., 1983.
4. Ciarcia, S. "Build An Unlimited-Vocabulary Speech Synthesizer," Byte, October 1982.
5. Davies, Russ. COMPUTES!'s Mapping the IBM PC and PCjr, COMPUTE! Publications, Inc., 1985.
6. El-Imam, Yousif A. "A Personal Computer-based Speech Analysis and Synthesis System," IEEE Micro, June 1987.
7. Elovitz, Honey S., Rodney Johnson, Astrid McHugh, and John E. Shore. "Letter-to-Sound Rules for Automatic Translation of English Text to Phonetics," IEEE Transactions on Acoustics, Speech, and Signal Processing, 1976.
8. Fry, Dennis. Homo Loquens - Man As A Talking Animal, Cambridge University Press, 1977.
9. Heid, Jim. "The Echo Voice Synthesizer," Microcomputing, April 1984.
10. Holmes, J. N. M & B Monographs - Speech Synthesis, Mills & Boon Limited, 1972.
11. International Business Machines Corporation, A Guide to Writing BASIC Speech Programs for the IBM PCjr, 1984.
12. Klatt, Dennis H. "Review of Text-to-Speech Conversion for English," Journal of the Acoustical Society of America, September 1987.
13. Kucera, Henry, and W. Nelson Francis, Computational Analysis of Present-Day American English, Brown University Press, 1967.
14. Lawrence, J. S. "Smoothtalker," Computers & The Humanities, April-June 1986.

15. Linggard, R. Electronic Synthesis of Speech, Cambridge University Press, 1985.
16. Morris, H. M. "Conversing With Computers & Robots," Control Engineering, August 1986.
17. Norton, Peter. Programmer's Guide to the IBM PC, Microsoft Press, 1985.
18. Omotayo, O. R. "Converting Text Into Speech in Real Time With Microcomputers," Microprocessors & Microsystems, November 1984.
19. Stoffel, David M. "Human Factors in Multi-Media Systems Access: Translating Text Documents To Speech," IEEE Electronics and Aerospace Systems Conference, Washington, D.C., November 16-19, 1981.
20. Teja, Edward R. Teaching Your Computer To Talk, TAB Books, 1981.
21. Turbo Pascal, Version 3.01A, Borland International, 1985.
22. Vocal Interface Division, SC-01 Speech Synthesizer Data Sheet, Votrax, 1980.
23. Winston, Patrick Henry. Artificial Intelligence, Addison-Wesley Publishing Company, Inc., 1984.
24. Witten, Ian H. Principles of Computer Speech, Academic Press, Inc., 1982.
25. Wood, Steve. Using Turbo Pascal, Osborne McGraw-Hill, Inc., 1986.
26. Zenith Educational Systems, ET-18 Robot Voice Dictionary, Heath Company, 1983.
27. Zenith Educational Systems, HERO Robot User's Manual, Heath Company, 1983.

Appendix A

Letter-to-Sound Rules Used in the HEROTALK System

These are the rules loaded into memory, and used in the conversion of our English text to phonemes of the IPA (International Phonetic Association). The rules are stored in the format:

$$A[B]C = D$$

which translates as string "B" of text, preceded by string "A" and followed by string "C," is translated as string "D" in IPA. These rules are based on a set of translation rules developed by Elovitz, Johnson, McHugh, and Shore, and were published in the article "Letter-to-Sound Rules for Automatic Translation of English Text to Phonetics" in the December 1976 issue of "IEEE Transactions On Acoustics, Speech, and Signal Processing." I have taken their set, have made minor modifications to it, and have added several "new" rules to handle various translation exceptions as well as rules to convert numbers and various punctuation symbols.

The various symbols used in the rules are defined as follows:

- "+" : one of "E," "I," or "Y"
- ":" : zero (0) or more consonants
- "&" : one of "S," "C," "G," "Z," "X," "J," "CH," or "SH"
- "@" : one of "T," "S," "R," "D," "L," "Z," "N," "J,"
"TH," "CH," or "SH"
- "^" : one consonant
- "#" : one or more vowels
- "." : one of "B," "D," "V," "G," "J," "L," "M," "N,"
"R," "W," or "Z"

"%" : one of "ER," "ES," "ED," "ING," or "ELY

A Rules

/ [A .]=/EY/[A] =/AX/ [AS] =/AE Z/ [ALWAYS] =/AO L W EY Z/
[ARE] =/AA R/ [AR]O=/AX R/[AR]#=/EH R/ ^[AS]#=/EY S/[A]WA
=/AX/[AW]=/AO/ :[ANY]=/EH N IY/[A]^+ #=/EY/#:[ALLY]=/AX L IY/
[AL]#=/AX L/[AGAIN]=/AX G EH N/#:[AG]E=/IH JH/[A]^+ : #=/AE/
:[A]^+ =/EY/[A]^%=/EY/ [ARR]=/AX R/[ARR]=/AE R/ :[AR] =/AA
R/[AR] =/ER/[AR]=/AA R/[AIR]=/EH R/[AI]=/EY/[AY]=/EY/[AU]=
/AO/#:[AL] =/AX L/#:[ALS] =/AX L Z/[ALK]=/AO K/[AL]^=/AO L/
:[ABLE]=/EY B AX L/[ABLE]=/AX B AX L/[ANG]+=/EY N JH/[A]
=/AE/|

B Rules

/ [B .]=/B IY/[BEEN] =/B IH N/ [BE]^#=/B IH/[BEING]=/B IY IH
NX/ [BOTH] =/B OW TH/ [BUS]#=/B IH Z/[BUIL]=/B IH L/ [BEAR]
=/B EY R/[BB]=/B/[B]=/B/|

C Rules

/ [C .]=/S IY/ [CH]^=/K/^E[CH]=/K/[CH]=/CH/ S[CI]#=/S AY/
[CI]A=/SH/[CI]O=/SH/[CI]EN=/SH/[C]+=/S/[CK]=/K/[COM]%=
/K AH M/[C]=/K/|

D Rules

/ [D .]=/D IY/[DONE] =/D AH N/ [DR .]=/D AA K T ER/#:[DED]
=/D IH D/.E[D] =/D/#^:E[D] =/T/ [DE]^#=/D IH/ [DO] =/D UW/
[DOES]=/D AH Z/ [DOING]=/D UW IH NX/ [DOW]=/D AW/[DU]A=/JH
UW/ [DIE] =/D AY/[DD]=/D/[D]=/D/|

E Rules

/ [E .]=/IY/ [ELEVEN] =/IY L EH V AX N/ [ENABLE]=/EH N EY B
AX L/ [ENGINE]=/EH N JH IH N/ [ETC .]=/EH T S EH T ER AA/
[EVENING] =/IY V N IH NX/[EVERYONE]=/EH V ER IY W AA N/#:[E]

=/ /' ^:[E] =/ / :[E] =/IY/#[ED] =/D/#:[E]D =/ / [EV]ER=/EH
V/[E]^%=/IY/[ERI]#=/IY R IY/[ERI]=/EH R IH/#:[ER]#=/ER/
[ER]#=/EH R/[ER]=/ER/ [EVEN]=/IY V EH N/#:[E]W=/ /@[EW]=
/UW/[EW]=/Y UW/[E]O=/IY/#:&[ES] =/IH Z/#:[E]S =/ /#:[ELY] =
/L IY/#:[EMENT]=/M EH N T/[EFUL]=/F UH L/[EE]=/IY/[EARN]
=/ER N/ [EAR]^=/ER/ [EAD]=/EH D/#:[EA] =/IY AX/[EA]SU=/EH/
[EA]=/IY/[EIGH]=/EY/[EI]=/IY/ [EYE]=/AY/[EY]=/IY/[EU]=/Y UW/
[E]=/EH/|

F Rules

/ [F .]=/EH F/[FINALLY] =/F AY N AX L IY/ [FRIDAY]=/F R AY D
EY/[FUL]= /F UH L/[FRIEND]=/F R EH N D/[FINISH]=/F IH N IH
SH/[FF]=/F/[F]=/F/|

G Rules

/ [G .]=/JH IY/ [GAS]=/G AE S/[GIV]=/G IH V/ [G]I^=/G/[GE]T=
/G EH/SU[GGES]=/G JH EH S/[GG]=/G/ B#[G]=/G/[G]+=/JH/[GREAT]=
/G R EY T/#[GH]=/ / [G]=/G/|

H Rules

/ [H .]=/EY CH/[HEARD] =/HH ER D/ [HAV]=/HH AE V/ [HERE]=/HH
IY R/ [HOUR]=/AW ER/[HOW]=/HH AW/ [HEY] =/HH EY/ [HI] =/HH
AY/[H]#=/HH/[H]=/ /|

I Rules

/ [I .]=/AY/[IMPLY] =/IH M P L AY/ [IS] =/IH Z/[ISLAND]=/AY L
AH N D/ [IN]=/IH N/ [I] =/AY/[IN]D=/AY N/[IER]=/IY ER/#:R
[IED] =/IY D/[IED] = /AY D/[IEN]=/IY EH N/[IE]T=/AY EH/ :[I]
%=/AY/[I]%=/IY/[IE]=/IY/[I]^+:#=/IH/[IR]#=/AY R/[IZ]%=/AY Z/
[IS]%=/AY Z/[I]D%=/AY/+^[I]^+=/IH/[I]T%=/AY/#:[I]^+=/IH/[I]
^+=/AY/[IR]=/ER/[IGH]=/AY/[ILD]= /AY L D/[IGN] =/AY N/[IGN]
^=/AY N/[IGN]%=/AY N/[IQUE]=/IY K/ [I'M] =/AY M/[I'LL]=/AY
L/[I]=/IH/|

J Rules

/ [J .]=/JH EY/[JULY] =/JH UW L AY/[J]=/JH/|

K Rules

/ [K .]=/K EY/ [K]N=/ / [K]=/K/|

L Rules

/ [L .]=/EH L/ [LIVE] =/L IH V/ [LIVED] =/L IH V D/ [LIVING]
=/L IH V IH NX/[LO]C#=/L OW/L[L]=/ /#^:[L]%=/AX L/[LEAD]=
/L IY D/[LL]=/L/[L]=/L/|

M Rules

/ [M .]=/EH M/[MAYBE] =/M EY B IY/ [MEANT] =/M EH N T/
[MOREOVER] =/M OW R OW V ER/ [MR .]=/M IH S T ER/ [MRS .]=
/M IH S AX S/ [MS .] =/M IH Z/[MOV]=/M UW V/[MM]=/M/[M]
=/M/|

N Rules

/ [N .]=/EH N/[NATURE] =/N EY CH ER/ [NONE] =/N AH N/
[NOWHERE] = /N OW W EY R/E[NG]+=/N JH/[NG]R=/NX G/[NG]#=/NX
G/[NGL]%=/NX G AX L/[NG]=/NX/[NK]=/NX K/ [NOW] =/N AW/[N]=
/N/|

O Rules

/ [O .]=/OW/[OF] =/AX V/[OROUGH]=/ER OW/#:[OR] =/ER/#:[ORS]
=/ER Z/[OR]=/AO R/ [ONE]=/W AH N/[OW]=/OW/ [OVER]=/OW V
ER/[OV]=/AH V/[O]^%=/OW/[O]^EN=/OW/[O]^I#=/OW/[OL]D=/OW L/
[OUGHT]=/AO T/[OUGH]=/AH F/ [OU]=/AW/H[OU]S#=/AW/[OUS]=/AX
S/[OUR]=/OW R/[OULD]=/UH D/ [OU]^L=/AH/[OUP]=/UW P/[OU]=/AW/
[OY]=/OY/[OING]=/OW IH NX/[OI]=/OY/[OOR]=/AO R/[OOK]=/UH
K/[OOD]=/UH D/[OO]=/UW/[O]E=/OW/[O] =/OW/[OA]=/OW/ [ONLY]=

/OW N L IY/ [ONCE]=/W AH N S/[ON'T]=/OW N T/C[O]N=/AA/[O]NG=
 /AO/ ^:[O]N=/AH/I[ON]=/AX N/#:[ON] =/AX N/#^[ON]=/AX N/[O]
 ST =/OW/[OF]^=/AO F/[OTHER]=/AH DH ER/[OSS] =/AO S/#^[OM]
 =/AH M/ [OH] =/OW/[O]=/AA/|

P Rules

/ [P .]=/P IY/[PH]=/F/[PEOP]=/P IY P/[POW]=/P AW/[PUT] =/P UH
 T/ [PURPOSE]=/P ER P AX S/[PROGRAM]=/P R OW G R AE M/[PP]=
 /P/[P]=/P/|

Q Rules

/ [Q .]=/K Y UW/[QUAR]=/K W AO R/[QU]=/K W/[Q]=/K/|

R Rules

/ [R .]=/AA R/ [RECORD]=/R EH K AO R D/ [RE]^#=/R IY/[ROBOT]
 =/R OW B AA T/[RR]=/R/[R]=/R/|

S Rules

/ [S .]=/EH S/ [SAYS] =/S EH Z/[SHALL]=/SH AE L/[SEVEN]=/S EH
 V AX N/[SOMEONE]=/S AH M W AH N/[SHOE]=/SH UW/[SH]=/SH/
 #[SION]=/ZH AX N/[SOME]=/S AH M/#[SUR]#=/ZH ER/[SUR]#=/SH ER/
 #[SU]#=/ZH UW/#[SSU]#=/SH UW/#[SED] =/Z D/#[S]#=/Z/[SAID]
 =/S EH D/^[SION]=/SH AX N/[S]S=/ /. [S] =/Z/#:.E[S] =/Z/#^:##
 [S] =/Z/#^:#[S] =/S/U[S] =/S/ :#[S] =/Z/ [SCH]=/S K/[S]C+=/ /
 #[SM]=/Z M/#[SN]'=/Z AX N/'[S]=/Z/[S]=/S/|

T Rules

/ [T .]=/T IY/[THEREBY]=/DH EH R B AY/[TUESDAY]=/T UW Z D EY/
 [THE] = /DH AX/[TO] =/T UW/[THAT] =/DH AE T/ [THIS] =/DH IH
 S/ [THEY]=/DH EY/ [THERE]=/DH EH R/[THER]=/DH ER/[THEIR]=/DH
 EH R/ [THAN] =/DH AE N/ [THEM] =/DH EH M/[THESE] =/DH IY Z/
 [THEN]=/DH EH N/[THROUGH]=/TH R UW/[THOSE]=/DH OW Z/[THOUGH]
 =/DH OW/ [THUS]=/DH AH S/[TH]=/TH/#:[TED] =/T IH D/S[TI]
 #N=/CH/[TI]O=/SH/[TI]A=/SH/[TIEN]=/SH AX N/ [TUR]#=/CH ER/
 [TU]A=/CH UW/ [TWO]=/T UW/[TIE]=/T AY/[TT]=/T/[T]=/T/|

U Rules

/ [U .]=/Y UW/ [UN]I=/Y UW N/ [UN]=/AH N/ [UPON]=/AX P AA N/
@[UR]#= /UH R/[UR]#=/Y UH R/[UR]=/ER/[U]^=/AH/[U]^=/AH/[UY]
=/AY/ G[U]#=/ / G[U]%=/ /G[U]#=/W/#N[U]=/Y UW/@[U]=/UW/[U]=/Y
UW/|

V Rules

/ [V .]=/V IY/[VIEW]=/V Y UW/[V]=/V/|

W Rules

/ [W .]=/D AH B AX L Y UW/[WEDNESDAY]=/W EH N Z D EY/[WOMAN]
=/W UH M AX N/[WOMEN]=/W IH M AX N/ [WERE]=/W ER/[WA]S=/W
AA/[WA]T=/W AA/[WHERE]=/WH EH R/[WHAT]=/WH AA T/[WHOL]=/HH
OW L/[WHO]=/HH UW/[WH]=/WH/[WAR]=/W AO R/[WOR]^=/W ER/[WR]
=/R/ [WANT]=/W AA N T/[W]=/W/|

X Rules

/ [X .]=/EH K ~ S/[X]=/K ~ S/|

Y Rules

/ [Y .]=/W AY/[YOUNG]=/Y AH NX/ [YOUR]=/Y OW R/ [YOU]=/Y UW/
[YES] =/Y EH S/ [Y]=/Y/#^:[Y] =/IY/#^:[Y]I=/IY/ :[Y] =/AY/
:[Y]#=/AY/ :[Y]^+:#=/IH/ :[Y]^#=/AY/#:[Y]=/IY/[Y]=/IH/|

Z Rules

/ [Z .]=/Z IY/[Z]=/Z/|

Numerical Digit Rules

/[1]=/W AH N/[2]=/T UW/[3]=/TH R IY/[4]=/F OW R/[5]=/F AY V/
[6]=/S IH K ~ S/[7]=/S EH V AX N/[8]=/EY T/[9]=/N AY N/[0]=
/Z IY R OW/|

Punctuation Symbol Rules

/[&]=/AE N D/[@]=/AE T/[#]=/N AH M B ER/[%]=/P ER S EH N T/
[+]=/P L AH S/[=]=/IY K W UH L Z/[*]=/T AY M Z/[\$]=/D AA L
ER Z/|

Appendix B

S-Record Output Format

The S-record format for output modules was devised for the purpose of encoding programs or data files in a printable format for transportation between computer systems. When viewed by the user, S-Records are essentially character strings comprised of several fields identifying each record's record type, record length, memory address, code/data, and checksum. Each field is made up of bytes of binary data encoded as 2-character hexadecimal numbers; the first character represents the high-order 4 bits, and the second the low-order 4 bits of the byte.

The five fields which comprise an S-Record are shown below:

Type	Record Length	Address	Code/Data	Checksum
------	---------------	---------	-----------	----------

where the fields are composed as follows:

<u>Field</u>	<u>Characters</u>	<u>Contents</u>
Type	2	S-Record type - S0, S1, etc.
Record Length	2	The count of the character pairs in the record, excluding the type and record length.
Address	2	The 2-byte address at which the code/data, in the code/data field, is to be loaded into memory.
Code/Data	0 - 2n	From 0 to n bytes of executable code, memory-loadable data, or descriptive information.

Checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.
----------	---	--

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

S-Record Types

Eight types of S-Records have been defined to accommodate the several needs of the encoding, transportation, and decoding functions. In the case of the HERO robot, and the information needed to be transferred from HEROTALK to the robot, three types are of importance. They are:

S0 - The header record for each block of S-Records. The code/data field may contain any descriptive information identifying the soon to follow block of S-Records.

S1 - A record containing code/data and the 2-byte address at which the code/data is to reside at the remote site.

S9 - A termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is to be passed.

Normally, only one header and one termination record is used for each block of S-Records. Shown below is a typical S-Record format module as it would appear ready for data transfer:

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The module consists of one S0 record, four S1 records, and an S9 record. The S0 record is comprised of the following character pairs:

- S0 - S-Record type S0, indicating that it is a header record.
- 06 - Hexadecimal 06 (decimal 6), indicating that six character pairs (or ASCII bytes) follow.
- 00 - Four-character 2-byte address field, zeroes in this 00 example.
- 48
- 44 - ASCII H, D, and R, which denote "HDR".
- 52
- 1B - The checksum.

The first S1 record is explained as follows:

- S1 - S-Record type S1, indicating that it is a code/data record to be loaded at a 2-byte address.
- 13 - Hexadecimal 13 (decimal 19), indicating that 19 character pairs, representing 19 bytes of binary data, follow.

- 00 - Four-character 2-byte address field; hexadecimal 00, address 0000; where the data which follows is to be loaded.

The next 16 character pairs of the first S1 record are the ASCII bytes of the actual data. In the case of the HEROTALK system, these 16 character pairs would probably represent 16 speech phonemes which are to be spoken by the HERO robot. Each character pair would represent the hexadecimal equivalent of a Votrax phoneme.

- 2A - The checksum of the first S1 record.

The second and third S1 records each also contain \$13 (19 decimal) character pairs and end with checksums 13 and 52, respectively. The fourth S1 record contains 07 character pairs and has a checksum of 92.

The S9 record is explained as follows:

- S9 - S-Record type S9, indicating that it is a termination record.
- 03 - Hexadecimal 03, indicating that three character pairs (3 bytes) follow.
- 00 - The address field, zeroes in this case.
00
- FC - The checksum of the S9 record.

Appendix C

HEROTALK Operation Guide

HEROTALK is a microcomputer-based text-to-speech phoneme translator which enables the user to type in English text and have that text converted to speech phonemes usable by the Votrax SC-01 speech synthesizer built into the Heathkit ET-18 Robot (HERO) to produce speech. HEROTALK runs on the IBM-PC or compatible.

Overview

Though the program is for the most part self-explanatory, a few points merit attention:

ENTERING TEXT:

To enter English text, you may either prepare a file of text externally (using an editor of your choice) and have HEROTALK read text from that file, or you may enter English text directly from the keyboard. An options screen will appear at the point when you are to enter text, and at that time you may choose your desired option. It is important that if you have prepared a text file for HEROTALK to read, be very careful to check that text for errors. HEROTALK will read in your text exactly as you have typed it and thus, any errors in that text will be read in as well. In addition, DO NOT give HEROTALK a file of "gibberish" as your English text. HEROTALK will process the text, and produce the appropriate translations, but those translations will not be of any value since

HERO speaks only English.

CREATED FILES:

Upon one complete execution of HEROTALK, four files are created. These four files have the extensions "IPA," "HRO," "HEX," and "REC," and correspond to the IPA, Votrax, and HEX translations of your previously entered English text, as well as the "S-Records" constructed from your text required to enable the robot to speak.

IPA: The IPA translation is your English text converted into the speech phonemes of the IPA, or International Phonetic Association. This translation is saved to enable you to convert English text to a standard set of phonemes, and thus the opportunity to use HEROTALK in conjunction with other speech synthesis units.

HRO: The HRO translation is your English text converted into the speech phonemes particular to the Heathkit ET-18 Robot (i.e., Votrax phonemes). This translation is saved as a record of the Votrax phonemes, particular to the HERO robot, used to speak your entered text.

HEX: The HEX translation is your English text converted to its hexadecimal representation (i.e., the hex-

adecimal codes which correspond to the previously mentioned Votrax phonemes -- those phonemes particular to the HERO robot). This translation is saved to enable you to convert English text, and later use that translation (i.e., the hexadecimal codes) in another program written for the robot.

REC: The "S-Records" are a set of data records constructed to enable data to be transmitted between the IBM-PC and the HERO robot. These "S-Records" consist of character strings made up of several fields (record type, length, memory address, code/data, and checksum) and are used to accurately send data from the PC to the robot. As you do not need to fully understand the "S-Record" format to run the program, it will not be covered here extensively. For more information, consult the source code of the HEROTALK program or the description of the "S-Record" format (Appendix B).

When these files are created, you will be prompted for a name for each. The extensions "IPA," "HRO," "HEX," and "REC" will be appended automatically depending on which file is being created. A default name will always be offered to you, and you may choose it instead of choosing your own. BE CAREFUL, however, that you do not overwrite a previously created file that you wish to keep.

One final note ... Your PC MUST be configured with at least 256K of RAM in order to use HEROTALK. Configurations of 320K or more are recommended, but you must have at least 256K.

Input to the HEROTALK System

The text-to-speech process is initiated by the input of a sample of English text. Currently, you may enter this text in one of two ways:

F - Read From Text File

You may, using an editor of your choice, prepare a passage of text prior to running HEROTALK, and have HEROTALK read your text directly from that previously prepared file. Keep in mind that the passage should be checked for errors as HEROTALK will process this passage exactly as it is stored in this file. When using a text file to enter your English text to HEROTALK, you may also give the name of that file as a parameter when invoking HEROTALK. For example, if the file containing your English text is named "TEST.DAT," you can invoke HEROTALK with the command:

```
A>HEROTALK TEST.DAT
```

and HEROTALK will automatically read English text from the file TEST.DAT. Finally, when preparing a text file of data, be careful that the editor you use does not

insert additional control characters, etc., into the text.

K - Read From Keyboard

You may also enter your passage of text directly from the keyboard. Use this method if you do not have your passage saved in an external file. Using the keyboard, merely type in the text you wish to have converted into speech phonemes. For correcting mistakes you may use the backspace key, though once you move to a new line text on the previous line is no longer available for changes. This means that you should make sure each line is "correct" before hitting a carriage return and advancing to the next line of text.

When entering your text the characters you type WILL wrap around to the next line. However, the program DOES NOT necessarily "wrap" characters at the end of a word. Consequently, if you let the program "wrap" characters there is a very strong chance that a line will be broken in the middle of a word. A better approach is to use the carriage return as you approach the right margin, and continue typing characters on the next line. Whichever approach you choose, the program will function correctly.

AVAILABLE FUNCTION KEYS WHEN ENTERING TEXT VIA THE KEYBOARD

F1 - DONE :

Use this key ONLY when you are through entering your text and are ready to continue with the translation process. Upon hitting this key, your text will be converted into IPA phonemes, and subsequently to a set transferable to the HERO robot.

F2 - START OVER :

Use this key in the event you are dissatisfied with the text you have entered thus far and wish to start over. This key can be pressed at any time during the input of English text via the keyboard. KEEP IN MIND, HOWEVER, THAT THIS ERASES ALL TEXT YOU HAVE ENTERED TO THIS POINT! Be careful!

F3 - HELP :

Use this key to bring up the help screen.

In addition to text input you may also, in the event you have a word which you feel will not be translated correctly, enter a translation rule to ensure that the word in question IS translated correctly. To use this feature, choose:

A - Add New Translation Rule

Finally, two additional choices exist:

H - Help : Informative Help

E - Exit : Exit from the HEROTALK program, and return to
DOS

Entering New Rules During Execution

HEROTALK permits you to define "new" and "specific" translation rules for the "English-to-IPA" translation. If, in the English text you pass to HEROTALK a particular word is not translated correctly, you may define a translation rule to insure that in future executions of HEROTALK the word IS translated correctly. This allows the user to further broaden the range of HEROTALK's translation capabilities, and gives HEROTALK the ability to translate virtually ANY word the user desires.

When entering English text via the keyboard you may enter new translation rules PRIOR to entering your English text. This is particularly useful in the event that you plan to use words in your text which you suspect HEROTALK will have difficulty translating -- uncommon English words, and abbreviations for example. By defining new rules for these suspect words prior to actually executing HEROTALK, your speech translation will be much more accurate, and in the end more intelligible.

A new translation rule is composed of two parts: an English word, and its IPA (International Phonetic Association) translation.

ENGLISH PART:

The English part of a rule is merely the English representation of the word, abbreviation, acronym, etc., you wish to add to HEROTALK's rule set. When entering new rules, HEROTALK allows each new rule to translate only a SINGLE word. Consequently, no spaces are allowed in the English part of a new rule. If you need to add a new rule which will translate a phrase containing spaces, each sub-part of this phrase must be defined as a separate rule. For example, to enter a new rule for the "word" "IBM PC," you will need to enter first a rule for "IBM" and next a rule for "PC."

IPA PART:

The IPA part of a rule is the phonetic sound(s) which represent(s) the English "word" you entered in the aforementioned "English Part." It is very important that you enter the EXACT phonemes which represent the English word in question. Remember, all words corresponding to the English text you entered in the "English Part" will translate into the phonemes you enter in the "IPA Part." Consequently, you must be very careful when entering the IPA translation for a given English word. When entering new rules, an IPA table is provided in the upper half of the screen to help you derive

correct IPA translations.

Examples:

English part - COMPUTER : IPA part - K AH M P Y UW T ER

English part - JOHN : IPA part - JH AA N

Some notes to keep in mind ...

New translation rules are not permanently saved by HEROTALK; they exist only so long as the user does not leave HEROTALK, and return to the DOS level. Consequently, if new rules are defined during an execution of HEROTALK, and the user then chooses to quit, and return to DOS, those rules will no longer be defined when HEROTALK is executed again later. If the user wishes to have those rules "in effect," he will need to re-enter them. On the other hand, once a new rule is defined, it need not be redefined for continuous text translations within the same execution of HEROTALK (i.e., choosing the "CONTINUE" option after one complete text translation). Once HEROTALK is executed, all "new" translation rules added during that execution are saved until that execution is terminated, and the user returns to DOS.

Finally, in addition to this feature HEROTALK also has an "IPA Editor" which allows the user to edit HEROTALK's IPA translation of the entered English text. In the event that the new rule you are entering is NOT a word which you plan to use often, you may be better off letting HEROTALK attempt to translate it, and then, if necessary, edit this translation

later.

At this point, the computer has converted your "English Text" to IPA phonemes, and awaits your next command.

Your choices are:

F1 - EDIT : The program has an editor which will allow you to edit the IPA translation of your "English Text." Choosing this option will place you into a screen editor, enabling you to make modifications to the computer's IPA translation. The editor can be a valuable asset in improving the accuracy of the IPA translation. However, it can also be very damaging if you are not careful.

F2 - CONTINUE : Upon choosing this option, your IPA phonemes will be translated into Votrax phonemes, and upon "readying" the robot to receive data, sent to it enabling speech.

F3 - ADD TEXT : Choose this option if you wish to add additional "English Text" prior to proceeding with the translation process. All "English Text" entered prior to this will be saved, and uploaded later with the new text you wish to enter.

F4 - QUIT : Upon choosing this option, the IPA translation of your "English Text" will be saved in an external file, and the program will stop.

F5 - HELP : This HELP screen.

Using The IPA Editor

The IPA editor has been added as an aid for helping the user ensure that his IPA translation is as accurate as possible. By using it, a user may change any of the IPA phonemes which correspond to the translation of his previously entered "English Text," and thus almost guarantee intelligible and accurate speech from any "English Text" entered. The editor can be a great help in "fine-tuning" the speech translation process, but it can also be a disaster if not used with GREAT CAUTION! If you do not feel comfortable using the editor, PLEASE DO NOT!!!

SOME IMPORTANT NOTES ABOUT THE EDITOR

As mentioned above, the IPA editor is designed primarily as a small, "no-frills" editor which enables the user to make minor corrections/additions to the IPA translation generated by the program. For this reason the editor has some characteristics which merit emphasis:

Inserting Text: When inserting text, the editor WILL NOT automatically wrap text from one line to the next in instances where the line being added to exceeds the right margin. In fact, if, while inserting text, the line being

added to reaches the right margin, the program will "beep" and not allow further text insertion. In most cases the editor leaves enough "blank space" at the end of each line to allow the user to add additional text, thus no serious problems should occur. In the event you need more blank space than is available, use the "F3" key to insert a new line and complete the insertion on the "open" line. Remember: the editor was designed for making SMALL corrections to the program's IPA translation -- not MAJOR additions to it! If you wish to append text to the END of the IPA translation ("END" referring to the VERY end of the current IPA translation) you will save yourself a great deal of confusion if you insert your "text to be appended" BEFORE the last "*" in the IPA translation. The program will function correctly if you do not, but because text inserted after the last "*" in the editor is viewed as text not related to your previously entered "English Text," the editor will treat it as such, and thus not display it upon exiting the editor. This can be confusing and thus, it is not a good idea to try to append additional IPA phonemes to the current IPA translation which are not additions, or changes to what was originally translated by the program from your "English Text." If you wish to add MORE text, choose option "F3" (Enter More Text) upon leaving the editor.

UNDOing Changes : The UNDO key (F2) enables you to edit a

given line in the IPA translation, and then restore it to its original state should you decide you were happier with the line in its original state than in its current edited state. This is a good key to remember in the event you edit a line, and inadvertently edit it into disarray. However, it is important to note that the UNDO key only works with the line currently being edited. In other words, if you edit line 1 and then move the cursor down to line 2, you cannot go back to line 1 and restore it (i.e., line 1) to its original state. However, you may use the UNDO key as many times as you like while on a given line. Thus, you can edit and restore a line over and over so long as you remain on that line.

Aborting Edit : The QUIT key (F9) will exit the editor AT ANY TIME and restore the IPA translation to its original state (i.e., the state it was in upon entering the IPA editor). Thus, if you are editing the IPA translation and decide you have destroyed it, hit the F9 key and you should be all right. This is not to be confused with the SAVE key (F10) which will overwrite the old IPA translation with the new EDITED translation. Be careful!

Blank Spaces : As a final note, please pay CLOSE attention to where blanks occur in the IPA translation as they are very important in converting from IPA phonemes to the other phoneme sets used in the program. This becomes extremely

important when you insert and delete text from the IPA translation. When inserting text, be very careful to insert blanks where needed. Use the existing IPA translation as a guide.

AVAILABLE FUNCTION KEYS WHEN USING THE IPA EDITOR

- F1 - HELP : Will bring up this help screen.
- F2 - UNDO : Will "UNDO" all changes made to a line during the editing process.
- F3 - LNIN : Inserts a line at the current cursor position.
- F4 - LNDE : Deletes the line at the current cursor position.
- F5 - BLKL : Will move the cursor to the left, one block of IPA phonemes at a time.
- F6 - BLKR : Will move the cursor to the right, one block of IPA phonemes at a time.
- F7 - TOP : Will position the cursor at the top of the IPA list currently being edited.
- F8 - END : Will position the cursor at the bottom of the IPA list currently being edited.
- F9 - QUIT : Will abort the edit, leaving the IPA list exactly as it was upon entering the editor. A good key to remember!
- F0 - SAVE : Will exit the editor, saving the new, edited IPA list. Use ONLY if you wish to replace the IPA list with the new edited list.

Additional Keys Available

- Backspace : Will backspace the cursor to the left one character at a time, deleting the character to the left of the current position.
- Del : Will delete the character at the current cursor position, and shift the remaining characters all one character to the left.
- Home : Will position the cursor at the beginning of the current line.
- End : Will position the cursor at the end of the current line.
- PgUp : Will page the IPA list up one page (i.e., 6 lines).
- PgDn : Will page the IPA list down one page (i.e., 6 lines).
- Return : Moves the cursor down one line.
- UpArrow : Moves the cursor up one line.
- DownArrow : Moves the cursor down one line.
- RightArrow : Moves the cursor to the right one character.
- LeftArrow : Moves the cursor to the left one character.
- Ins : Toggles INSERT mode. When in INSERT mode, the cursor will appear as a "large block". When in OVERWRITE mode, the cursor will be as normal.

Sending Data to the HERO Robot

This is the final step in the "text-to-speech" translation process and will send speech phonemes, corresponding to the

"English Text" you previously entered, to the HERO robot. At the conclusion of this step the robot will be ready to speak.

To send your translated "English Text" to HERO you must "ready" it to receive data. Thus, the following instructions must be carried out before proceeding with the "upload":

- 1) Check to make sure a cable connects the COM1 serial port of the IBM-PC you are using with the serial interface on the HERO robot.
- 2) Check to make sure the dip switches on the HERO robot are set for 9600 baud.
- 3) Turn the HERO robot on (after turning the robot on, it will respond with the word "READY").
- 4) On the robot's keypad, press the "RESET" key (again, the robot will respond with the word "READY").
- 5) Finally, on the robot's keypad press the keys "3" and "A."

After completing the above instructions, you are ready to upload to the robot. Press key "F1" to begin the upload. When the upload has completed, the robot will respond with the word "READY."

To hear your text spoken by the robot:

- 1) Press the "RESET" key on the robot (again, it will respond with the word "READY").
- 2) Press the keys "A," "D," and "0040" on the robot's keypad. Upon pressing the final "0," the robot will begin to speak the English text you entered at the beginning of the program.

NOTE : After the robot finishes speaking your text you must press the "RESET" key and the keys "3" and "A," on the robot's keypad, before uploading additional text in subsequent program executions.